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 21206

Балашова Вячеслава Вадимовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
А. Ю. Власенко

Новосибирск 2022

Содержание

Цель

1. Знакомство с архитектурой x86/x86-64
2. 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. 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;
- (опционально) сравнить различия в программах для архитектуры x86 и архитектуры x86-64.

4. Составить отчет по лабораторной работе. Отчет должен содержать следующее:

- Титульный лист.
- Цель лабораторной работы.
- Полный компилируемый листинг реализованной программы на Си.
- Листинги на ассемблере (O0 и O3).
- Подробное описание найденных оптимизаций, примененных на уровне O3.
- Вывод по результатам лабораторной работы.

Описание работы

Ход выполнения работы:

1. Был создан файл main.cpp с исходным кодом программы на языке программирования C++ для вычисления функции sin(x) с помощью разложения этой функции в ряд Маклорена по первым N членам:

$$\sin(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} = x - \frac{\boldsymbol{x^3}}{\boldsymbol{3!}} + \frac{\boldsymbol{x^5}}{\boldsymbol{5!}} - \frac{\boldsymbol{x^7}}{\boldsymbol{7!}} + \cdots + \frac{\boldsymbol{(-1)^{n-1}}}{\boldsymbol{(2n-1)!}} \boldsymbol{x^{2n-1}} + \cdots;$$

$$\boldsymbol{-\infty < x < +\infty}$$

2. На сайте godbolt.org была произведена генерация ассемблерного для синтаксиса AT&T с двумя уровнями оптимизации: -O0 и -O3.

3. Был проведен анализ полученного листинга и исходного кода с добавлением комментариев в листинги (Приложение 2 и Приложение 3).

4. Были сопоставлены команды языка C++ с машинными командами, определено размещение переменных в регистрах и ячейках памяти.

5. Проведены сравнения листингов уровней оптимизации компилятора O0 и O3 с поиском различий и оптимизаций, выполненных компилятором.

Заключение

В ходе практической работы была изучена архитектура x86/x86-64. Был выполнен анализ сгенерированных на сайте godbolt.org листингов для уровней оптимизации компилятора O0 и O3 программы, вычисляющей функцию sin(x) через разложение в ряд, на языке C++. Были изучены основные команды языка ассемблера. Были сопоставлены команды языка C++ с машинными командами. Был проведен анализ размещения переменных в языке ассемблера, анализ отличий скомпилированных кодов на разных уровнях оптимизаций.

В ходе анализа отличий в листингах с уровнями оптимизации O0 и O3 были выявлены следующие оптимизации на уровне O3:

1. Задействуется гораздо больше регистров процессора (Например, регистры %R12, %RDX, %RCX)
2. Задействуются регистры меньших размеров (Например, чаще используются регистры %Exx, используются %R12d, %R12b)
3. Нестандартное использование регистров (%RBP в функции Sin() используется не для указания на базу кадра стека, а для хранения переменной n)
4. Отсутствие явно выделенных границ локального стека у всех функций (Не всегда %RSP указывает на вершину локального стека, а %RBP на базу кадра локального стека)
5. Замена функции atoi на функцию strtoll
6. Уменьшено количество обращений в оперативную память, что связано со скоростью работы оперативной памятью относительно скорости регистровой памяти (регистровая память гораздо быстрее оперативной)
7. Большинство переменных не имеют копий в оперативной памяти.
8. Оперативная память в основном используется для передачи 16-байтовых аргументов в функции и для конвертирования переменных из целочисленного типа в тип с плавающей точкой и наоборот с последующим перемещением на соответствующий регистр
9. Отсутствуют бесполезные арифметические операции (Например, смещение вершины стека на 16 байт, чтоб выкинуть аргументы после вызова функции, и сразу перед вызовом новой функции опять эти 16 байт под аргументы выделить)
10. Объект std::endl помещается в оператор вывода
11. Более эффективная схема условных/безусловных переходов
12. Inline подстановка функций где это уместно
13. Смена структур функций с целью оптимизации (Например, если функция power в качестве первого аргумента имеет число -1, то она преобразуется в более

эффективную версию, состоящую из счетчика и функции смены знака, функция fabs заменяется на команду для регистра)

14. Используется быстрое обнуление регистра с помощью команды xor

Приложение 1. Исходный код программы

```
#include <cmath>
#include <ctime>
#include <iostream>

const double pi = 3.14159265358979323846;

using namespace std;

long double power(long double x, size_t n)
{
   long double forRet = 1;
   for (size_t i = 1; i <= n; i++)
   {
      forRet *= x;
   }

   return forRet;
}

long double factorial(size_t n)
{
   long double forRet = 1;
   for (size_t i = 1; i <= n; i++)
   {
      forRet *= i;
   }

   return forRet;
}

long double Sin(long double x, size_t n)
{
   int sign = (x < 0) ? -1 : 1;
   x = fmod(fabs(x),2 * pi);
   if (x > pi)
   {
      x -= pi;
      sign *= -1;
   }
   if (x > pi/2)
   {
      x = pi - x;
   }

   long double result = 0;
   for (size_t i = 1; i <= n; i++)
   {
      long double buf = power(-1, i - 1) * power(x, 2 * i - 1) / factorial(2 * i - 1);
      if (isnan(buf))
      {
         result += 0;
```

```
        }
        else
        {
            result += buf;
        }
    }

    return result * sign;
}

int main(int argc, char* argv[])
{
    size_t numberOfElements = atoll(argv[1]);
    long double x = 10;

    cout << "Answer: " << Sin(x, numberOfElements) << endl;
    cout << "Time: "   << (long double)(clock()) / CLOCKS_PER_SEC << "s" << endl;

    return 0;
}
```

Приложение 2. Ассемблерный листинг (оптимизация O0)

```
power(long double, unsigned long):
        pushq   %rbp                    // Сохраняет указатель кадра вызвавшей подпрограммы
        movq    %rsp, %rbp              // Формирует базу стека локальных переменных
        movq    %rdi, -40(%rbp)         // Помещает значение регистра %rdi в оперативную память по адресу
(%rbp)-40 (n)
        fld1                            // Записывает в стек сопроцессора число 1 (forRet)
        fstpt   -16(%rbp)               // Записывает в оперативную память по адресу (%rbp)-16 число из вершины
стека сопроцессора (forRet)
        movq    $1, -24(%rbp)           // Записывает в оперативную память по адресу (%rbp)-24 число 1 (i)
        jmp     .L8                     // Перескакивает на метку .L8
.L9:
        fldt    -16(%rbp)               // Загружает в стек сопроцессора значение из оперативной памяти (%rbp)-
16 (forRet)
        fldt    16(%rbp)                // Загружает в стек сопроцессора значение из оперативной памяти
(%rbp)+16 (x)
        fmulp   %st, %st(1)             // Перемножает значение в регистре %st с значением в регистре %st(1),
помещает его в регистр %st(1), выталкивает вершину, делая %sp(1) - вершиной (forRet * x)
        fstpt   -16(%rbp)               // Помещает значение из вершины стека сопроцессора в оперативную
память по адресу (%rbp)-16 (forRet = forRet * x)
        addq    $1, -24(%rbp)           // Прибавляет единицу к значению в оперативной память по адресу (%rbp)-
24 (i++)
.L8:
        movq    -24(%rbp), %rax         // Помещает в регистр %rax значение из оперативной памяти по адресу
(%rbp)-24 (i)
        cmpq    %rax, -40(%rbp)         // Сравнивает значение в регистре %rax и значение из оперативной памяти
по адресу (%rbp)-40 (i и n)
        jnb     .L9                     // Если i <= n, то переходит на метку .L9
        fldt    -16(%rbp)               // Загружает в стек сопроцессора значение из оперативной памяти (%rbp)-
16 (forRet)
        popq    %rbp                    // Возвращает указатель кадра
        ret                             // Возврат из функции
factorial(unsigned long):
        pushq   %rbp                    // Сохраняет указатель кадра вызвавшей подпрограммы
        movq    %rsp, %rbp              // Формирует базу стека локальных переменных
        movq    %rdi, -40(%rbp)         // Помещает значение из регистра %rdi в оперативную память по адресу
(%rbp)-40 (n)
        fld1                            // Загружает число 1 в стек сопроцессора (forRet)
        fstpt   -16(%rbp)               // Загружает число 1 из стека сопроцессора в оперативную память по адресу
(%rbp)-16 и выталкивает число из стека сопроцессора (forRet)
        movq    $1, -24(%rbp)           // Помещает число 1 по адресу (%rbp)-24 (i)
        jmp     .L12                    // Прыгает на метку .L12
.L14:
        fildq   -24(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-24 в стек
сопроцессора (long double i)
        cmpq    $0, -24(%rbp)           // Сравнивает копию этого числа из оперативной памяти по адресу (%rbp)-
24 с нулем (i и 0)
        jns     .L13                    // Если флаг знака SF = 0, то переходит на метку .L13
        fldt    .LC2(%rip)              // Загружает в стек сопроцессора число по метке .LC2 (Предпроцессорная
магическая константа)
        faddp   %st, %st(1)             // Складывает числа в регистрах %st и %st(1), результат помещает в %st(1),
выталкивает %st
.L13:
        fldt    -16(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-16 в стек
сопроцессора (forRet)
        fmulp   %st, %st(1)             // Перемножает числа из стека сопроцессора %st и %st(1), помещает
результат в %st(1) и выталкивает %st (forRet * i)
```

```
    fstpt   -16(%rbp)               // Загружает число из стека сопроцессора в оперативную память по адресу (%rbp)-16 (forRet *= i)
    addq    $1, -24(%rbp)           // Прибавляет единицу к значению в оперативной памяти по адресу (%rbp)-24 (i++)
.L12:
    movq    -24(%rbp), %rax         // Помещает число из оперативной памяти по адресу (%rbp)-24 в регистр %rax (i)
    cmpq    %rax, -40(%rbp)         // Сравнивает число из регистра %rax со значением в стеке по адресу (%rbp)-40 (i и n)
    jnb     .L14                    // Если i <= n, то отправляется на метку .L14
    fldt    -16(%rbp)               // Загружает значение из оперативной памяти по адресу (%rbp)-16 в стек сопроцессора (forRet)
    popq    %rbp                    // Возвращает значение %rbp
    ret                             // Выходит из функции
Sin(long double, unsigned long):
    pushq   %rbp                    // Сохраняет указатель кадра вызвавшей подпрограммы
    movq    %rsp, %rbp              // Формирует базу стека локальных переменных
    subq    $96, %rsp               // Выделяет 96 байт
    movq    %rdi, -72(%rbp)         // Помещает значение регистра %rdi в стек по адресу (%rbp)-72 (n)
    fldt    16(%rbp)                // Загружает в стек сопроцессора второй аргумент подпрограммы (long double x)
    fldz                            // Загружает в стек константу +0.0
    fcomip  %st(1), %st             // Сравнивает значение x с нулем с установкой EFLAGS, выталкивает ноль
    fstp    %st(0)                  // Очищает вершину стека (выталкивает число 0)
    jbe     .L32                    // Если x >= 0, то перходит на метку .L32
    movl    $-1, %eax               // Помещает в регистр %eax число -1
    jmp     .L19                    // Переходит на метку .L19
.L32:
    movl    $1, %eax                // Помещает в регистр %rax число 1
.L19:
    movl    %eax, -4(%rbp)          // Помещает значение регистра %eax в стек по адресу (%rbp)-4 (sign)
    pushq   24(%rbp)                // Помещает на вершину стека первую половину числа x
    pushq   16(%rbp)                // Помещает на вершину стека вторую половину числа x
    call    std::fabs(long double)  // Вызывает функцию std::fabs для long double
    addq    $16, %rsp               // Затирает скопированное значение x на вершине стека
    movq    .LC4(%rip), %rax        // Помещает в регистр %rax число, находящееся по метке .LC4 (2 * pi)
    leaq    -16(%rsp), %rsp         // Выделяет 16 байт на вершине стека
    fstpt   (%rsp)                  // Помещает в выделенные 16 байт число из вершины стека сопроцессора - результат функции std::fabs()
    movq    %rax, %xmm0             // Копирует значение из регистра %rax в регистр %xmm0

    call    __gnu_cxx::__promote_2<decltype (((__gnu_cxx::__promote_2<long double, std::__is_integer<long double>::__value>::__type)(0))+((__gnu_cxx::__promote_2<double, std::__is_integer<double>::__value>::__type)(0))), std::__is_integer<decltype (((__gnu_cxx::__promote_2<long double, std::__is_integer<long double>::__value>::__type)(0))+((__gnu_cxx::__promote_2<double, std::__is_integer<double>::__value>::__type)(0)))>::__value>::__type std::fmod<long double, double>(long double, double)
                                    // Вызывает функцию fmod() от двух аргументов

    addq    $16, %rsp               // "Затирает" скопированное значение fabs(x) на вершине стека
    fstpt   16(%rbp)                // Записывает число из вершины стека сопроцессора в стек оперативной памяти по адресу (%rbp)+16 и удаляет число из стека сопроцессора (x = fmod())
    fldt    .LC5(%rip)              // Записывает вещественное число по метке .LC5 в вершину стека сопроцессора (long double pi)
    fldt    16(%rbp)                // Записывает вещественное число из стека оперативной памяти в вершину стека сопроцессора (x = fmod())
    fcomip  %st(1), %st             // Сравнивает значение числа pi и x с установкой EFLAGS, выталкивает x
    fstp    %st(0)                  // Очищает вершину стека сопроцессора
    jbe     .L20                    // Если pi >= x, то переходит на метку .L20
```

```
	fldt	16(%rbp)		// Записывает вещественное число из стека оперативной памяти в вершину стека сопроцессора (x = fmod())
	fldt	.LC5(%rip)		// Записывает вещественное число по метке .LC5 в вершину стека сопроцессора (long double pi)
	fsubrp	%st, %st(1)		// Вычитает значение %st из значения %st(1), запоминает его в %st(1), выталкивает %st, делая %st(1) - вершиной стека (x - pi)
	fstpt	16(%rbp)		// Записывает на место x = fmod() новое значени x из стека сопроцессора, удаляя значение из стека сопроцессора (x -= pi)
	negl	-4(%rbp)		// Меняет знак у числа sign
.L20:
	fldt	.LC6(%rip)		// Записывает вещественное число по метке .LC6 в вершину стека сопроцессора (long double pi/2)
	fldt	16(%rbp)		// Записывает вещественное число из стека оперативной памяти в вершину стека сопроцессора (x)
	fcomip	%st(1), %st		// Сравнивает значение числа x с pi/2 с установкой EFLAGS, выталкивает x
	fstp	%st(0)				// Очищает стек сопроцессора
	jbe	.L22			// Если x <= pi/2, то переходит на метку .L22
	fldt	.LC5(%rip)		// Записывает вещественное число по метке .LC5 в вершину стека сопроцессора (long double pi)
	fldt	16(%rbp)		// Записывает вещественное число из стека оперативной памяти в вершину стека сопроцессора (x = fmod())
	fsubrp	%st, %st(1)		// Вычитает значение %st из значения %st(1), запоминает его в %st(1), выталкивает %st, делая %st(1) - вершиной стека (pi - x)
	fstpt	16(%rbp)		// Записывает на место x новое значени x из стека сопроцессора, удаляя значение из стека сопроцессора (x)
.L22:
	fldz				// Загружает в стек сопроцессора число 0
	fstpt	-32(%rbp)		// Записывает вещественное число из вершины стека сопроцессора в стек оперативной памяти по адресу (%rbp)-32, выталкивает число из стека сопроцессора (result)
	movq	$1, -40(%rbp)		// Записывает в Оперативную память число 1 по адресу (%rbp)-40 (i)
	jmp	.L24			// Перескакивает на метку .L24
.L27:
	movq	-40(%rbp), %rax		// Загружает в регистр %rax значение из оперативной памяти по адресу (%rbp)-40 (i)
	subq	$1, %rax		// Вычитает число 1 из значения регистра %rax (i - 1)
	fld1				// Записывает в стек сопроцессора число 1
	fchs				// Меняет знак у вершины стека сопроцессора
	leaq	-16(%rsp), %rsp		// Выделяет 16 байт на вершине стека
	fstpt	(%rsp)			// Записывает число из вершины стека сопроцессора в вершину стека оперативной памяти
	movq	%rax, %rdi		// Копирует значение регистра %rax в регистр %rdi (i - 1)

	call	power(long double, unsigned long)
				// Вызывает функцию power от двух аргументов

	addq	$16, %rsp		// "Зачищает" выделенную память
	fstpt	-96(%rbp)		// Записывает вещественное число из вершины стека сопроцессора и выталкивает его (power)
	movq	-40(%rbp), %rax		// Помещает значение из стека по адресу (%rbp)-40 в регистр %rax (i)
	addq	%rax, %rax		// Удваивает значение в %rax (2 * i)
	subq	$1, %rax		// Отнимает единицу из значения в %rax (2 * i - 1)
	pushq	24(%rbp)		// Помещает на вершину стека значение по адресу (%rbp)+24 (x)
	pushq	16(%rbp)		// Помещает на вершину стека значение по адресу (%rbp)+24 (x)
	movq	%rax, %rdi		// Копирует значение из регистра %rax в регистр %rdi (2 * i - 1)

	call	power(long double, unsigned long)
				// Вызывает функцию power от двух переменных
```

```
    addq    $16, %rsp               // Зачищает выделенную память на вершине стека, которая использовалась для передачи аргументов в функцию
    fldt    -96(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-96 в стек сопроцессора (power(-1))
    fmulp   %st, %st(1)             // Умножает значение в регистре %st на значение в регистре %st(1), сохраняет значение в регистре %st(1) и выталкивает значение %st
    fstpt   -96(%rbp)               // Загружает значение из стека сопроцессора в стек оперативной памяти по адресу (%rbp)-96 (power * power)
    movq    -40(%rbp), %rax         // Помещает значение из оперативной памяти по адресу (%rbp)-40 в регистр %rax (i)
    addq    %rax, %rax              // Удваивает значение в регистре %rax (2 * i)
    subq    $1, %rax                // Отнимает из значения в регистре %rax число 1 (2 * i - 1)
    movq    %rax, %rdi              // Копирует значение регистра %rax в регистр %rdi (2 * i - 1)
    call    factorial(unsigned long)// Вызывает функцию factorial от одной переменной
    fldt    -96(%rbp)               // Помещает значение из оперативной памяти по адресу (%rbp)-96 в стек сопроцессора (power * power)
    fdivp   %st, %st(1)             // Делит значение в регистре %st(1) на значение в регистре %st, записывает результат в регистре %st(1) и выталкивает значение %st (buf)
    fstpt   -64(%rbp)               // Записывает число из стека сопроцессора в оперативную память по адрессу (%rbp)-64, после чего выталкивает его из стека сопроцессора (buf)
    pushq   -56(%rbp)               // Помещает на вершину стека первую половину long double (buf)
    pushq   -64(%rbp)               // Помещает на вершину стека вторую половину long double (buf)
    call    std::isnan(long double) // Вызывает функцию std::isnan от одного аргумента
    addq    $16, %rsp               // Зачищает 16 байт на вершине стека
    testb   %al, %al                // Проверяет число в регистре %al
    je      .L25                    // Отправляется на метку .L25 если флаг ZF = 1
    fldt    -32(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-32 в стек сопроцессора (result)
    fldz                            // Загружает число 0 в стек сопроцессора
    faddp   %st, %st(1)             // Складывает значение в регистре %st со значением в регистре %st(1), помещает значение в регистр %st(1), выталкивает значение %st (result += 0)
    fstpt   -32(%rbp)               // Загружает значение из стека сопроцессора в стек оперативной памяти по адресу (%rbp)-32 (result += 0)
    jmp     .L26                    // Безусловный переход на метку .L26
.L25:
    fldt    -32(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-32 в стек сопроцессора (result)
    fldt    -64(%rbp)               // Загружает число из оперативной памяти по адресу (%rbp)-64 в стек сопроцессора (buf)
    faddp   %st, %st(1)             // Складывает значение в регистре %st со значением в регистре %st(1), помещает значение в регистр %st(1), выталкивает значение %st (result += buf)
    fstpt   -32(%rbp)               // Загружает значение из стека сопроцессора в стек оперативной памяти по адресу (%rbp)-32 (result += buf)
.L26:
    addq    $1, -40(%rbp)           // Прибавляет к значению в оперативной памяти по адресу (%rbp)-40 число 1 (i++)
.L24:
    movq    -40(%rbp), %rax         // Помещает число из стека по адрресу (%rbp)-40 в регистр %rax (i)
    cmpq    %rax, -72(%rbp)         // Сравнивает число в регистре %rax и в оперативной памяти по адресу rbp-72 (i и n)
    jnb     .L27                    // Если i <= n, то переходит на метку .L27
    fildl   -4(%rbp)                // Загружает в стек сопроцессора целое число из оперативной памяти по адресу (%rbp)-4 (sign)
    fldt    -32(%rbp)               // Загружает в стек сопроцессора число из оперативной памяти по адресу (%rbp)-32 (result)
    fmulp   %st, %st(1)             // Перемножает числа из стека сопроцессора %st и %st(1), помещает результат в %st(1) и выталкивает %st (result * sign)
    leave                           // Возвращает значения кадра и стека
    ret                             // Выход из функции
```

```
.LC9: // Метки с константными строками
	.string "Answer: "
.LC10:
	.string "Time: "
.LC12:
	.string "s"
main:
	pushq   %rbp                // Помещает значение регистра %rbp в стек, сохраняет указатель кадра вызвашей программы
	movq    %rsp, %rbp          // Копирует значение регистра %rsp в регистр %rbp, формирует начало кадра локальных переменных
	pushq   %rbx                // Помещает значение регистра %rbx в стек, так как в дальнейшем регистр будет использоваться
	subq    $72, %rsp           // Отнимает из значения регистра %rsp число 72, "выделяя" 72 байта на стеке
	movl    %edi, -52(%rbp)     // Копирует значение регистра %edi в стек по адресу (%rbp)-52 (argc)
	movq    %rsi, -64(%rbp)     // Копирует значение регистра %rsi в стек по адресу (%rbp)-64 (argv[][])
	movq    -64(%rbp), %rax     // Копирует значение из стека в регистр %rax (argv[][])
	addq    $8, %rax            // Прибавляет к значению регистра %rax число 8 (&argv[1][])
	movq    (%rax), %rax        // Переходит по адресу, хранящемуся в %rax, и записывает значение по этому адресу в %rax (argv[1][])
	movq    %rax, %rdi          // Копирует значение регистра %rax в регистр %rdi (argv[1][])
	call    atoll               // Запоминает адрес следующей команды (movq) и вызывает функцию atoll, после выполнения которой переходит к следующей за ней (movq)
                                // Вызывает функцию, переводящую строку в 8-байтовое число, и возвращает его через регистр %rax
	movq    %rax, -24(%rbp)     // Копирует значение регистра %rax в стек по адресу (%rbp)-24 (numberOfElements = atoll)
	fldt    .LC8(%rip)          // Загружает вещественное число по метке .LC8 на вершину стека сопроцессора (long double x = 10)
	fstpt   -48(%rbp)           // Записывает вещественное число в стек оперативной памяти и «выталкивает» его из стека сопроцессора (x)
	movl    $.LC9, %esi         // помещает адрес метки со строкой “Answer: “ в регистр %esi (“Answer: “)
	movl    $_ZSt4cout, %edi    // помещает адрес объекта cout в регистр %edi (cout)

	call    std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*)
                                // Запоминает адрес следующей команды (movq) и вызывает оператор <<, после выполнения которого переходит к следующей за оператором функции (movq)
                                // Выводит строку по метке .LC9 (“Answer: “)

	movq    %rax, %rbx          // Копирует значение регистра %rax в регистр %rbx (результат оператора <<)
	movq    -24(%rbp), %rax     // Копирует значение из оперативной памяти (numberOfElements) в регистр %rax
	pushq   -40(%rbp)           // Копирует на вершину стека первую половину значения long double x
	pushq   -48(%rbp)           // Копирует на вершину стека вторую половину значения long double x
	movq    %rax, %rdi          // Копирует в регистр %rdi значение регистра %rax (numberOfElements)

	call    Sin(long double, unsigned long)
                                // Сохраняет адрес функции addq в стек, вызывает функцию Sin(), после чего возвращается к функции addq, в дальнейшем запоминание функции, к которой нужно вернуться, считаем по умолчанию выполненным для функции call

	addq    $16, %rsp           // Выкидывает верхние 16 байт из стека
	leaq    -16(%rsp), %rsp     // Выделяет 16 байт на вершине стека (помещает в %rsp адрес (%rsp) – 16)
	fstpt   (%rsp)              // Записывает в вершину стека оперативной памяти число с вершины стека сопроцессора и выталкивает это число с вершины стека сопроцессора (Sin())
	movq    %rbx, %rdi          // Копирует значение из регистра %rbx в регистр %rdi (cout)
```

```
    call    std::basic_ostream<char, std::char_traits<char> >::operator<<(long double)
                                // Выводит long double Sin()

    addq    $16, %rsp           // "Затирает" 16 байт, в которых хранился long double Sin()

    movl    $_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_, %esi (std::endl)
                                // Перемещает адрес функции std::endl в регистр %esi

    movq    %rax, %rdi          // Копирует значение регистра %rax в регистр %rdi (cout <<)

    call    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char,
std::char_traits<char> >& (*)(std::basic_ostream<char, std::char_traits<char> >&))
                                // Вызывает оператор вывода в поток cout для std::endl;

    movl    $.LC10, %esi        // Помещает адрес метки со строкой “Time: “ в регистр %esi ("Time: ")
    movl    $_ZSt4cout, %edi    // Помещает в регистр %edi адрес объекта cout (cout)

    call    std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*)
                                // Выводит строку, адрес которой хранится в %esi

    movq    %rax, %rbx          // Копирует значение из регистра %rax в регистр %rbx (cout <<)
    call    clock               // Вызывает функцию, которая вернет количество квантов процессора,
прошедших с момента запуска подпрограммы
    movq    %rax, -72(%rbp)     // Помещает значение регистра %rax (результат функции clock()) по адресу
(%rbp)-72 (Результат clock())
    fildq   -72(%rbp)           // берет результат функции clock() и загружает в сопроцессор на вершину
стека    (Результат clock())
    fldt    .LC11(%rip)         // Загружает вещественное число по метке .LC11 в сопроцессор на вершину
стека (CLOCKS_PER_SEC)
    fdivrp  %st, %st(1)         // Делит значение в регистре %st(1) на значение в регистре %st, записывает
его в регистр %st(1), и выталкивает из стекового регистра %st, делая %st(1) - вершиной (clock() /
CLOCKS_PER_SEC)
    leaq    -16(%rsp), %rsp     // Выделяет 16 байт на стеке
    fstpt   (%rsp)              // помещает на вершину стека результат деления в сопроцессоре (clock() /
CLOCKS_PER_SEC)
    movq    %rbx, %rdi          // Копирует значение из регистра %rbx в регистр %rdi (cout <<)

    call    std::basic_ostream<char, std::char_traits<char> >::operator<<(long double)
                                // Вызывает оператор вывода long double

    addq    $16, %rsp           // Затирает только что выведенное значение
    movl    $.LC12, %esi        // Помещает адрес строки в регистр %esi ("s")
    movq    %rax, %rdi          // Помещает значение регистра %rax в регистр %rdi (cout <<)

    call    std::basic_ostream<char, std::char_traits<char> >& std::operator<<
<std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*)
                                // Вызывает оператор для вывода строк

    movl    $_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_, %esi (std::endl)
                                // Помещает адрес функции std::endl в регистр %esi

    movq    %rax, %rdi          // Помещает значение регистра %rax в регистр %rdi (cout <<)

    call    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char,
std::char_traits<char> >& (*)(std::basic_ostream<char, std::char_traits<char> >&))
                                // Вызывает оператор для вывода std::endl
```

```
    movl    $0, %eax                // Помещает 0 в регистр %rax
    movq    -8(%rbp), %rbx          // Восстанавливает значение %rbx
    leave                           // Эквивалентно movq %rbp %rsp; popq %rbp – восстанавливает значения кадра и стека до исходных состояний
    ret                             // Выходит из подпрограммы

__static_initialization_and_destruction_0(int, int):
    pushq   %rbp                    // Запоминает значение регистра %rbp – указатель кадра вызвавшей подпрограммы
    movq    %rsp, %rbp              // Помещает в регистр %rbp значение регистра %rsp, тем самым формируя указатель начало кадра локальных переменных
    subq    $16, %rsp               // Резервирует место из 16 байт в стеке
    movl    %edi, -4(%rbp)          // Копирует значение из регистра %edi в оперативную память
    movl    %esi, -8(%rbp)          // Копирует значение из регистра %edi в оперативную память
    cmpl    $1, -4(%rbp)            // Сравнивает значения 1 и значение в оперативной памяти по адресу (%rbp)-4
    jne     .L39                    // Если 1 != значение в оперативной памяти по адресу (%rbp)-4, то переходит на метку .L39
    cmpl    $65535, -8(%rbp)        // Сравнивает значения 65535 и значение в оперативной памяти по адресу (%rbp)-8
    jne     .L39                    // Если 65535 != значение в оперативной памяти по адресу (%rbp)-8, то переходит на метку .L39
    movl    $_ZStL8__ioinit, %edi
    call    std::ios_base::Init::Init() [complete object constructor]
    movl    $__dso_handle, %edx
    movl    $_ZStL8__ioinit, %esi
    movl    $_ZNSt8ios_base4InitD1Ev, %edi
    call    __cxa_atexit
.L39:
    nop                             // Нет операции
    leave                           // Восстанавливает значение кадра и стека
    ret                             // Выходит из функции
_GLOBAL__sub_I_power(long double, unsigned long):
    pushq   %rbp                    // Запоминает значение регистра %rbp – указатель кадра вызвавшей подпрограммы
    movq    %rsp, %rbp              // Помещает в регистр %rbp значение регистра %rsp, тем самым формируя указатель начало кадра локальных переменных
    movl    $65535, %esi            // Записывает в регистр %esi число 65535
    movl    $1, %edi                // Записывает в регистр %edi число 1

    call    __static_initialization_and_destruction_0(int, int)
                                    // Вызывает функцию от двух переменных

    popq    %rbp                    // Возвращает указатель кадра
    ret                             // Выходит из функции
.LC2: // 18446744073709551616.0000000
    .long   0
    .long   -2147483648
    .long   16447
    .long   0
.LC4: // double 2 * pi
    .long   1413754136
    .long   1075388923
.LC5: // long double pi
    .long   560513024
    .long   -921707870
    .long   16384
    .long   0
.LC6: // long double pi/2
```

```
        .long   560513024
        .long   -921707870
        .long   16383
        .long   0
.LC8:     // long double x = 10;
        .long   0
        .long   -1610612736
        .long   16386
        .long   0
.LC11: // CLOCKS_PER_SEC
        .long   0
        .long   -198967296
        .long   16402
        .long   0
```

Приложение 3. Ассемблерный листинг (оптимизация O3)

```
power(long double, unsigned long):
        fldt    8(%rsp)                 // Загружает в стек сопроцессора число из оперативной памяти (x)
        testq   %rdi, %rdi              // Проверяет значение с %rdi
        je      .L11                    // Если в %rdi хранится 0, то прыгает на метку .L11
        fld1                            // Загружает число 1 в стек сопроцессора (forRet)
        movl    $1, %eax                // Помещает число 1 в регистр %rax (i)
.L10:
        addq    $1, %rax                // Увеличивает значение в регистре %rax на 1 (i++)
        fmul    %st(1), %st             // Перемножает числа в регистрах %st и %st(1), помещает ответ в %st
(forRet *= x)
        cmpq    %rax, %rdi              // Сравнивает числа в регистрах %rax и %rdi (i <= n)
        jnb     .L10                    // Если i <= n, то отправляется по метке .L10
        fstp    %st(1)                  // Убирает ненужные элементы в стеке
        ret                             // Возвращение из функции
.L11:
        fstp    %st(0)                  // Убирает верхнее значение в стеке сопроцессора (return 1)
        fld1                            // Загружает число 1 на вершину стека сопроцессора
        ret                             // Возвращение из функции
factorial(unsigned long):
        testq   %rdi, %rdi              // Проверяет значение %rdi
        je      .L17                    // Если 0 == n, то переходит на метку .L17
        movl    $1, %eax                // Записывает число 1 в регистр %eax (i)
        fld1                            // Загружает в стек сопроцессора число 1 (forRet)
.L16:
        movq    %rax, -16(%rsp)         // Копирует значение регистра %rax в оперативную память (i)
        fildq   -16(%rsp)               // Загружает из оперативной памяти целое число в стек сопроцессора (i)
        testq   %rax, %rax              // Проверяет значение в регистре %rax
        jns     .L15                    // Если знак не сменился то отправляется на метку .L15
        fadds   .LC2(%rip)              // Прибавляет к значению в регистре %st число по метке .LC2 (константа,
известная на этапе компиляции)
.L15:
        fmulp   %st, %st(1)             // Перемножает числа в регистрах %st и %st(1) и запоминает значение
в %st(1), выталкивая %st
        addq    $1, %rax                // Увеличивает значение в регистре %rax на число 1 (i++)
        cmpq    %rax, %rdi              // Сравнивает значение в регистрах %rax и %rdi (i и n)
        jnb     .L16                    // Если i <= n, то переходит на метку .L16
        ret                             // Возвращается из функции
.L17:
        fld1                            // Помещает в стек сопроцессора число 1 (return 1)
        ret                             // Возвращается из функции
Sin(long double, unsigned long):
        pushq   %r12                    // Запоминает значение регистра %r12 в стеке
        movl    $1, %eax                // Помещает в регистр %eax число 1
        pushq   %rbp                    // Запоминает значение регистра %rbp в стеке (база кадра вызвавшей
программы)
        movq    %rdi, %rbp              // Копирует значение из регистра %rdi (n) в регистр %rbp
        pushq   %rbx                    // Запоминает значение регистра %rbx в стеке
        movl    $-1, %ebx               // Помещает число -1 в регистр %ebx
        subq    $16, %rsp               // Вычитает из значения регистра %rsp число 16, тем самым выделяя 16
байт на вергине стека
        fldt    48(%rsp)                // Загружает из оперативной памяти по адресу %rsp+48 в стек сопроцессора
        fldz                            // Загружает число 0 на вершину стека сопроцессора
        fcomip  %st(1), %st             // Сравнивает число в регистрах %st(1) (x) и %st (0) и выталкивает %st
        fldl    .LC4(%rip)              // Загружает вещественное число по метке .LC14 (2 * pi)
        cmovbe  %eax, %ebx              // Помещает 1 в %ebx, если x >= 0
```

```
    seta    %r12b                   // Устанавливает байт в регистр %r12b, если x < 0 (sign)
    fstpt   (%rsp)                  // Помещает число из стека сопроцессора в оперативную память (2 * pi)
    movq    (%rsp), %rax            // Помещает значение из оперативной памяти в регистр %rax (2 * pi)
    movl    8(%rsp), %edx           // Помещает значение из оперативной памяти в регистр %edx (2 * pi)
    pushq   %rdx                    // Помещает значение из %rdx на вершину стека (2 * pi)
    pushq   %rax                    // Помещает значение из %rax на вершину стека (2 * pi)
    fabs                            // Функция, высчитывающая модуль
    subq    $16, %rsp               // Выделяет память на вершине стека
    fstpt   (%rsp)                  // Помещает на вершину стека оперативной памяти число из стека
сопроцессора и выталкивает это число из стека сопроцессора (fabs(x))
    call    fmodl                   // Вызывает функцию fmod, ее результат лежит в стеке сопроцессора
    addq    $32, %rsp               // Зачищает выделенные 32 байта по аргументы функции
    fldl    .LC5(%rip)              // Загружает константу по метке .LC5 (pi) в стек сопроцессора
    fxch    %st(1)                  // Меняет местами регистры %st и %st(1) (aka swap(x, pi);
    fcomi   %st(1), %st             // Сравнивает значения в регистрах %st(1) и %st (pi и x)
    jbe     .L48                    // Если pi >= x, то прыгает на метку .L48
    movzbl  %r12b, %r12d            // Присваивает %r12d значение %r12b и расширяет нулевым значением
    fsubp   %st, %st(1)             // Вычитает из значения регистра %st (x) значение регистра %st(1) (pi),
сохраняет значение в регистре %st(1), выталкивает значение %st (x-=pi)
    leal    -1(%r12,%r12), %r12d    // Вычисляет адрес -1+%r12+%r12 и помещает его в %r12d
    movl    %r12d, %ebx             // Копирует значение из регистра %r12d в регистр %ebx (sign)
    jmp     .L22                    // Отправляется на метку .L22
.L48:
    fstp    %st(1)                  // Выкидывает значение pi из стека сопроцессора
.L22:
    fldl    .LC6(%rip)              // Загружает в стек сопроцессора число по метке .LC6 (pi/2)
    fxch    %st(1)                  // Меняет местами регистры %st и %st(1) (aka swap(pi/2, x);
    fcomi   %st(1), %st             // Сравнивает значения в регистрах %st(1) и %st (pi/2 и x)
    fstp    %st(1)                  // Выталкивает pi/2 из стека
    jbe     .L24                    // Если pi/2 >= x, то отправляется по метке .L24
    fsubrl  .LC5(%rip)              // Загружает вещественное число по метке .LC5, из него вычитает значение
в регистре %st, и помещает результат в регистр %st
.L24:
    testq   %rbp, %rbp              // Проверяет значение %rbp (n)
    je      .L36                    // Если n == 0, то отправляется на метку .L36
    fldz                            // Загружает число 0 в стек сопроцессора (result)
    xorl    %edx, %edx              // Зануляет значение в регистре %edx
    movl    $1, %ecx                // Помещает число 1 в регистр %ecx
    flds    .LC2(%rip)              // Загружает вещественное число по метке .LC2 (предрасчитанная
константа)
    testq   %rdx, %rdx              // Проверяет значение в регистр %rdx
    je      .L37                    // Если значение в %rdx == 0, то отправляется на метку .L37 (Всегда?)
.L47:
    movl    $1, %eax                // в регистр %eax помещается число 1
    fld1                            // в стек сопроцессора помещается число 1
.L28:
    fchs                            // Меняет знак у числа в регистре %st (-1)
    addq    $1, %rax                // Прибавляет число 1 к значению в регистре %rax
    cmpq    %rax, %rdx              // Сравнивает значения в регистрах %rax и %rdx
    jnb     .L28                    // Если значение в %rax <= значение в %rdx, то отправляется на метку .L28
(pow(-1, i-1)
    movl    $1, %eax                // Помещает в регистр %eax число 1
    fld1                            // Загружает в стек сопроцессора число 1
.L29:
    addq    $1, %rax                // Прибавляет к значению в регистре %rax число 1
    fmul    %st(4), %st             // Перемножает значения в регистрах %st(4)
    cmpq    %rax, %rcx              // Сравнивает значения в регистрах %rax и %rcx
    jnb     .L29                    // Если значение в %rax <= %rcx, то переходит на метку .L29
    movq    %rcx, %rsi              // Помещает значение регистра %rcx в регистр %rsi
```

```
.L27:
    fmulp   %st, %st(1)         // Перемножает значения в регистрах %st и %st(1), результат ложит
в %st(1), выталкивает %st
    movl    $1, %eax            // Помещает число 1 в регистр %eax
    fld1                        // Загружает 1 в стек сопроцессора
.L31:
    movq    %rax, (%rsp)        // Копирует значение из регистра %rax в вершину стека
    fildq   (%rsp)              // Загружает целое число в стек сопроцессора из вершины стека
оперативной памяти
    testq   %rax, %rax          // Проверяет значение %rax
    jns     .L30                // Если в %rax содержится положительное число, то прыгает на метку .L30
    fadd    %st(3), %st         // К значению в регистре %st прибавляется значение из регистра %st(3)
.L30:
    addq    $1, %rax            // В rax ложится число 1
    fmulp   %st, %st(1)         // Перемножает значения в регистрах %st и %st(1), результат ложит
в %st(1), выталкивает %st
    cmpq    %rax, %rsi          // Сравнивает значения в регистрах %rax и %rsi
    jnb     .L31                // Если значение в %rax <= начения в регистре %rsi, то отправляется на
метку .LC31
    fdivrp  %st, %st(1)         // Делит число в регистре %st(1) на число в регистре %st, записывает
результат в %st(1) и выталкивает %st
    fucomi  %st(0), %st         // Сравнивает значение в регистре %st и %st(???)
    jp      .L46                // Если значения в регистрах %st и %st одинаковые, то отправляется на
метку .L46
    faddp   %st, %st(2)         // Прибавляет к значению в регистре %st(2) значение в регистре %st и
.L33:
    leaq    1(%rdx), %rax       // Помещает в регистр %rax адрес 1+(%rdx)
    addq    $2, %rdx            // Прибавляет к значению в регистре %rdx число 2
    addq    $2, %rcx            // Прибавляет к значению в регистре %rcx число 2
    cmpq    %rdx, %rbp          // Сравнивает значения в регистрах %rdx и %rbp
    jb      .L49                // Если значения: %rdx > %rbp, то переходит по метке .L49
    movq    %rax, %rdx          // Копирует значение из регистра %rax в регистр %rbx
    testq   %rdx, %rdx          // Проверяет значение %rdx
    jne     .L47                // Если %rdx != 0, то отправляется на метку .L47
.L37:
    fld     %st(2)              // Загружает в стек сопроцессора значение из стека сопроцессора из
регистра %st(2)
    movl    $1, %esi            // Помещает число 1 в регистр %esi
    fld1                        // Загружает число 1 в стек сопроцессора
    fxch    %st(1)              // swap(%st, %st(1))
    jmp     .L27                // Отправляется на метку .L27
.L36:
    fstp    %st(0)              // Выталкивается значение из вершины стека сопроцессора
    fldz                        // В стек сопроцессора загружается число 0 (result)
    jmp     .L26                // Отправляется на метку .L26
.L49:
    fstp    %st(0)
    fstp    %st(1)
.L26:
    movl    %ebx, (%rsp)        // Помещает число из регистра %ebx в вершину стека (sign)
    fildl   (%rsp)              // Загружает из вершины стека оперативной памяти целое число в стек
сопроцессора (sign)
    addq    $16, %rsp           // Затирает верхние 16 байт стека
    popq    %rbx                // Восстанавливает значение %rbx
    popq    %rbp                // Восстанавливает значение %rbp
    popq    %r12                // Восстанавливает значение %r12
    fmulp   %st, %st(1)         // Перемножает числа в регистрах %st и %st(1), помещает результат
в %st(1), выталкивает %st (result * sign)
    ret                         // Выход из функции
```

```
.L46:
	fstp	%st(0)			// Удаляет верхнее значение из стека сопроцессора
	fldz				// Помещает в стек сопроцессора число 0
	faddp	%st, %st(2)		// Складывает значения в регистрах %st и %st(2), помещает результат
в %st(2), выталкивает %st
	jmp	.L33			// Прыгает на метку .L33

// Строковые константы
.LC8:
	.string "Answer: "
.LC10:
	.string "Time: "
.LC12:
	.string "s"
main:
	pushq	%rbx			// Сохраняет предыдущее значение регистра %rbx
	movl	$10, %edx		// Помещает число 10 в регистр %edx
	subq	$16, %rsp		// Выделяет 16 байт на вершине стека
	movq	8(%rsi), %rdi		// Копирует значение argv[1][] из регистра %rsi в регистр %rdi
	xorl	%esi, %esi		// Зануляет половину битов регистра %esi
	call	strtoll			// Вызывает функцию strtoll
	movl	$.LC8, %esi		// Помещает адрес метки со строкой
	movl	$8, %edx		// Помещает число 8 в регистр %edx
	movl	$_ZSt4cout, %edi	// Помещает адрес cout в регистр %edi
	movq	%rax, %rbx		// Копирует значение %rax (numberOfElements) в регистр %rbx

	call	std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<char,
std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*, long)
				// Вызывает оператор вывода << для строки "Answer: "

	flds	.LC9(%rip)		// Загружает число в стек сопроцессора из метки .LC9 long double 10
	movq	%rbx, %rdi		// Помещает значение из %rbx (numberOfElements) в регистр %rdi
	fstpt	(%rsp)			// Помещает число из регистра сопроцессора в оперативную память и
выталкивает его из стека сопроцессора (long double 10)
	movq	(%rsp), %rax		// Копирует значение из оперативной памяти в регистр %rax
	movl	8(%rsp), %edx		// Копирует еще 4 байта
	pushq	%rdx			// Помещает на вершину стека значение регистра %rdx
	pushq	%rax			// Помещает на вершину стека значение регистра %rax

	call	Sin(long double, unsigned long)
				// Вызывает функцию Sin()

	movl	$_ZSt4cout, %edi	// Помещает cout в %edi
	fstpt	(%rsp)			// Помещает число из стека сопроцессора на в оперативную память

	call	std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char,
std::char_traits<char> >::_M_insert<long double>(long double)
				// Вызывает оператор вывода cout << для long double Sin()

	movq	%rax, %rdi		// Помещает результат вывода в регистр %rdi
	popq	%rax			// Удаляет значение с вершины стека и помещает его в регистр %rax
	popq	%rdx			// Удаляет значение с вершины стека и помещает его в регистр %rdx

	call	std::basic_ostream<char, std::char_traits<char> >& std::endl<char,
std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&) [clone .isra.0]
				// Выводит std::endl

	movl	$6, %edx		// Помещает число 6 в регистр %edx
	movl	$.LC10, %esi		// Помещает адрес метки со строкой "Time: "
```

```
	movl	$_ZSt4cout, %edi	// Помещает cout в регистр %edi

	call	std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<char,
std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*, long)
					// Вызывает оператор вывода строки "Time: "

	call	clock			// Вызывает функцию clock()
	subq	$16, %rsp		// Выделяет 16 байт на вершине стека
	movl	$_ZSt4cout, %edi	// Помещает cout в %edi
	movq	%rax, 16(%rsp)		// Помещает результат функции clock() в оперативную память
	fildq	16(%rsp)		// Загружает целое число clock() в стек сопроцессора
	fdivs	.LC11(%rip)		// Делит clock() в стеке сопроцессора на вещественное число по
метке .LC11 (CLOCKS_PER_SEC) и помещает результат на место clock()
	fstpt	(%rsp)			// Помещает число clock() из вершины стека сопроцессора в оперативную
память, после чего выталкивает его из сопроцессора

	call	std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char,
std::char_traits<char> >::_M_insert<long double>(long double)
					// Выводит long double (clock() / CLOCKS_PER_SEC)

	popq	%rcx			// Помещает значение из вершины стека оперативной памяти в
регистр %rcx и удаляет его из вершины стека оперативной памяти
	movl	$1, %edx		// Помещает 1 к значению в регистре %edx
	popq	%rsi			// Помещает значение из вершины стека оперативной памяти в регистр %rsi
и удаляет его из вершины стека оперативной памяти
	movq	%rax, %rbx		// Помещает результат вывода cout << в регистр %rbx
	movl	$.LC12, %esi		// Загружает адрес метки .LC12 со строкой "s" в регистр %esi
	movq	%rax, %rdi		// Копирует значение cout из регистра %rax в регистр %rdi

	call	std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<char,
std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*, long)
					// Вызывает оператор вывода для строки "s"

	movq	%rbx, %rdi		// Помещает cout в регистр %rdi

	call	std::basic_ostream<char, std::char_traits<char> >& std::endl<char,
std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&) [clone .isra.0]
					// Вызывает оператор вывода для std::endl

	addq	$16, %rsp		// Прибавляет 16 к значению %rsp, затирая память в 16 байт
	xorl	%eax, %eax		// Зануляет регистр %eax
	popq	%rbx			// Возвращает значение %rbx
	ret				// Выходит из программы

_GLOBAL__sub_I_power(long double, unsigned long):
	subq	$8, %rsp
	movl	$_ZStL8__ioinit, %edi
	call	std::ios_base::Init::Init() [complete object constructor]
	movl	$__dso_handle, %edx
	movl	$_ZStL8__ioinit, %esi
	movl	$_ZNSt8ios_base4InitD1Ev, %edi
	addq	$8, %rsp
	jmp	__cxa_atexit

.LC2:
	.long	1602224128

.LC4:// 2 * pi
	.long	1413754136
```

```
        .long   1075388923

.LC5: // pi
        .long   1413754136
        .long   1074340347

.LC6: // pi / 2
        .long   1413754136
        .long   1073291771

.LC9: // long double 10
        .long   1092616192

.LC11:// CLOCKS_PER_SEC
        .long   1232348160
```