###### **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 №3**

ВВЕДЕНИЕ В АРХИТЕКТУРУ x86/x86-64

студента 2 курса, группы 23201

Сорокина Матвея Павловича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.С. Матвеев

Новосибирск 2024

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ** 3](#_Toc175333134)

[**ЗАДАНИЕ** 3](#_Toc175333135)

[**ОПИСАНИЕ РАБОТЫ** 4](#_Toc175333136)

[***Управление операций с памятью*** 4](#_Toc175333137)

[***Упрощение операций в FPU стеках*** 5](#_Toc175333138)

[**ЗАКЛЮЧЕНИЕ** 6](#_Toc175333139)

[**ПРИЛОЖЕНИЯ** 7](#_Toc175333140)

# **ЦЕЛЬ**

Знакомство с программной архитектурой x86/x86-64 и 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.

1. Для программы на языке Си (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры x86 и архитектуры x86-64, используя различные уровни комплексной оптимизации.
2. 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

1. Составить отчет, отражающий этапы работы, результаты анализа, выводы.

# **ОПИСАНИЕ РАБОТЫ**

1. Реализовано задание 2 - алгоритм вычисления числа Пи метом Монте-Карло на языке *C++*.

Время замерялось перед началом и после окончания работы функции, вычисляющей число Пи. Разность этих двух значений дает общее время выполнения функции. Для проверки точности измерений, код программы запускается несколько раз.

1. Ассемблерные листы с различными уровнями оптимизации генерировались с помощью:

* Компилятора:

|  |
| --- |
| g++ -m64 -S -o res\_x86-64 SinCalculation.cpp |
| g++ -m32 -S -o res\_x86 SinCalculation.cpp |
| g++ -S -masm=intel -o res\_x86-64 PiCalculation.cpp |

***-m64***: Указывает компилятору использовать 64-битную архитектуру. Следовательно на выход получим файл c 64-битным ассемблерным листингом (x86-64).

***-m32***: Указывает компилятору использовать 32-битную архитектуру.

***-masm=intel:*** Указывает компилятору использовать intel синтаксис.

* Сайта: <https://godbolt.org>

1. Взаимодействие *FPU* стека и стека вызовов на примере функции ***long double PiCalculation(long long n)***

Стек вызовов - фрейм стека вызовов создается при вызове функции, где размещаются локальные переменные и сохраненные регистры.

FPU стек - это отдельная структура данных, используемая сопроцессором для работы с числами с плавающей запятой.

Он содержит восемь регистров данных (data registers) от *st(0)* до *st(7),* организованных как стек (Last In, First Out).  В отличие от регистров общего назначения обычная пользовательская программа не может получить к ним прямой доступ.

Когда в ассемблерном листинге используются инструкции *FPU*,

такие как: ***fld***, ***fstp***, данные загружаются из обычного стека вызовов в *FPU* стек и обратно:

* инструкция ***fld TBYTE PTR [rbp-48]*** загружает значение переменной sign из стека вызовов (обычный стек) в вершину *FPU* стека *st(0).*
* инструкция ***fstp TBYTE PTR [rbp-48]*** выгружает значение из вершины *FPU* стека *st(0)* обратно в обычный стек вызовов в адрес [rbp-48], где хранятся переменные.

1. Оптимизация кода в ассемблерном листинге программы:

**Без оптимизации (-O0):**

1. **Использование стека**

Например, переменные *n* и *m* (хранятся по адресам *[rbp-40]* и *[rbp-8])* загружаются и сохраняются именно на стеке.

Промежуточные вычисления, такие как результат деления, тоже сохраняются в памяти, например, результат в *[rbp-48]:*

|  |
| --- |
| divsd   xmm0, xmm1  movsd   QWORD PTR [rbp-48], xmm0  fld     QWORD PTR [rbp-48] |

1. **Наличие избыточных операций**

Результат деления сохраняется сначала в *xmm0*, а затем сохраняется в память по адресу *[rbp-48],* после чего значение загружается в FPU регистр (инструкцией ***fld***), что создаёт накладные расходы.

**С оптимизацией (-O1):**

1. **Минимализация использования стека, использование регистра**

Переменные *n* и *m* теперь хранятся в регистрах процессора (*rbp* и *r12d* соответственно), что уменьшает количество обращений к памяти.

Промежуточные значения сохраняются в регистрах, только минимальные данные сохраняются на стеке (пример - результат деления сохраняется в *[rsp+8]).*

1. **Упрощение операций**

Использование FPU стека (регистров сопроцессора) сведено к минимуму, большинство арифметических операций выполняется напрямую в регистрах *xmm –* 128-битных регистрах векторных расширений.

1. Размещение переменных языка Си в программах на ассемблере на примере функции  ***long double PiCalculation(long long n)*** при уровне компиляции ***-O0***

В функции есть 4 переменные, каждая имеет свой адрес:

* *long long n* - параметр функции, передается при вызове.
* *long long m* - счётчик, увеличивающийся при попадании точки в единичный круг.
* *double x* – координата случайной точки по оси Ox.
* *double y* – коордианта случайной точки по оси Oy.

После вызова функции создается стандартный стековый фрейм (стек вызовов), где размещаются локальные переменные:

* n – хранится в стеке по адресу *[rbp-40]*
* *m* – хранится в стеке по адресу *[rbp-8]*
* *i –* счетчик цикла хранится по адресу *[rbp-16]*
* *x и y –* обрабатываются регистрами векторного расширения *xmm*

Как эти переменные хранятся в ассемблерном листинге:

|  |
| --- |
| PiCalculation(long long):  push rbp  mov rbp, rsp  sub rsp, 48  mov QWORD PTR [rbp-40], rdi ; параметр n сохраняется в [rbp-40]  mov QWORD PTR [rbp-8], 0 ; переменная m, инициализация в [rbp-8]  mov edi, 0  call time  mov edi, eax  call srand  mov QWORD PTR [rbp-16], 0 ; счётчик цикла i в [rbp-16]  jmp .L2  .L2:  mov rax, QWORD PTR [rbp-16] ; загружаем i в rax  cmp rax, QWORD PTR [rbp-40] ; сравниваем i с n  jl .L5  pxor xmm1, xmm1  cvtsi2sd xmm1, QWORD PTR [rbp-8] ; загружаем m в xmm1  movsd xmm0, QWORD PTR .LC2[rip]  mulsd xmm0, xmm ; умножаем 4.0 \* m  pxor xmm1, xmm1  cvtsi2sd xmm1, QWORD PTR [rbp-40] ; загружаем n в xmm1  divsd xmm0, xmm ; делим результат на n (4.0 \* m / n)  movsd QWORD PTR [rbp-48], xmm0 ; сохраняем результат в [rbp-48]  fld QWORD PTR [rbp-48] ; загружаем результат в FPU стек  leave  ret |

1. Используемое соглашение о вызовах (Calling convention)

**Соглашение о вызове** - описание технических особенностей вызова подпрограмм, определяющее:

* способы передачи параметров подпрограммам;
* способы вызова (передачи управления) подпрограмм;
* способы передачи результатов вычислений, выполненных подпрограммами, в точку вызова;
* способы возврата (передачи управления) из подпрограмм в точку вызова.

https://en.wikipedia.org/wiki/X86\_calling\_conventions

Основные признаки *System V AMD64 ABI* в моей программе:

1. Передача параметров:

* Первые шесть аргументов передаются через регистры: ***rdi, rsi, rdx, rcx, r8, r9*** (для целочисленных значений и указателей). Пример в коде - первый аргумент функции (типа *long long*, передаваемый через *rdi*) сохраняется в локальной переменной:

|  |
| --- |
| mov QWORD PTR [rbp-40], rdi |

* Первые восемь значений с плавающей точкой передаются через регситры: ***xmm0–xmm7***.

1. Возврат значений:

* Возврат целочисленных значений до 64 бит происходит через ***rax***, до 128 бит – через ***rdx***.
* Возврат чисел с плавающей точкой происходит через ***xmm0*** и ***xmm1***.

Пример: в конце функции (стр. 55-56) видно, что результат вычислений с плавающей точкой помещается в ***xmm0 –*** это подготовка к возврату значения:

|  |
| --- |
| movsd QWORD PTR [rbp-48], xmm0  fld QWORD PTR [rbp-48] |

1. Использование ***rbp*** регистра: используется как **base pointer**, чтобы отслеживать текущий фрейм стека.
2. Использование стека: локальные переменные хранятся в стеке, который создается инструкцией:

|  |
| --- |
| sub rsp, 48 |

# **ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы я познакомился с программной архитектурой x86/x86-64 и научился анализировать ассемблерный листинг программы для архитектуры x86/x86-64.

Без оптимизации компилятор стремится сохранить структуру исходного кода на языке программирования, что делает ассемблерный код более понятным, но менее эффективным в плане выполнения.

С оптимизацией компилятор активно использует доступные регистры для временного хранения и обработки данных, что существенно ускоряет выполнение программы.

# **ПРИЛОЖЕНИЯ**

**Приложение 1:Исходный код программы *PiCalculation.cpp***

|  |
| --- |
| #include <iostream>  #include <sys/times.h> // for times  #include <unistd.h> // for sysconf  #include <cstdlib> // for atoi and atof  long double PiCalculation(long long n) {      long long m = 0;      srand(time(NULL));      for (long long i = 0; i < n; i++) {          double x = static\_cast<double>(rand()) / RAND\_MAX \* 2.0 - 1.0;          double y = static\_cast<double>(rand()) / RAND\_MAX \* 2.0 - 1.0;          if (x \* x + y \* y <= 1.0) {              m++;          }      }      return 4.0 \* m / n;  }  int main(int argc, char \*argv[]) {      if (argc != 2) {          std::cerr << "Invalid number of arguments" << std::endl;          return 0;      }        struct tms start, end;      long clocks\_per\_sec = sysconf(\_SC\_CLK\_TCK);      long long n = atoll(argv[1]);      std::cout << "N = " << n << std::endl;      int runs = 5;      double min\_time = 10000;      for (long long i = 0; i < runs; i++) {          times(&start);          double pi = PiCalculation(n);          times(&end);          double taken\_time = static\_cast<double>(end.tms\_utime - start.tms\_utime) / static\_cast<double>(clocks\_per\_sec);          std::cout << "pi = " << pi << std::endl;          std::cout << "Run time: " << taken\_time << "\n" << std::endl;          min\_time = std::min(min\_time, taken\_time);      }      std::cout << "Min time: " << min\_time << " seconds" << std::endl;      return 0;  } |

**Приложение 2: ассемблерный листинг программы PiCalculation.cpp с ключом -O0, архитектурой x86-64**

|  |
| --- |
| PiCalculation(long long):          push    rbp          mov     rbp, rsp          sub     rsp, 48          mov     QWORD PTR [rbp-40], rdi          mov     QWORD PTR [rbp-8], 0          mov     edi, 0          call    time          mov     edi, eax          call    srand          mov     QWORD PTR [rbp-16], 0          jmp     .L2  .L5:          call    rand          pxor    xmm0, xmm0          cvtsi2sd        xmm0, eax          movsd   xmm1, QWORD PTR .LC0[rip]          divsd   xmm0, xmm1          addsd   xmm0, xmm0          movsd   xmm1, QWORD PTR .LC1[rip]          subsd   xmm0, xmm1          movsd   QWORD PTR [rbp-24], xmm0          call    rand          pxor    xmm0, xmm0          cvtsi2sd        xmm0, eax          movsd   xmm1, QWORD PTR .LC0[rip]          divsd   xmm0, xmm1          addsd   xmm0, xmm0          movsd   xmm1, QWORD PTR .LC1[rip]          subsd   xmm0, xmm1          movsd   QWORD PTR [rbp-32], xmm0          movsd   xmm0, QWORD PTR [rbp-24]          movapd  xmm1, xmm0          mulsd   xmm1, xmm0          movsd   xmm0, QWORD PTR [rbp-32]          mulsd   xmm0, xmm0          addsd   xmm1, xmm0          movsd   xmm0, QWORD PTR .LC1[rip]          comisd  xmm0, xmm1          jb      .L3          add     QWORD PTR [rbp-8], 1  .L3:          add     QWORD PTR [rbp-16], 1  .L2:          mov     rax, QWORD PTR [rbp-16]          cmp     rax, QWORD PTR [rbp-40]          jl      .L5          pxor    xmm1, xmm1          cvtsi2sd        xmm1, QWORD PTR [rbp-8]          movsd   xmm0, QWORD PTR .LC2[rip]          mulsd   xmm0, xmm1          pxor    xmm1, xmm1          cvtsi2sd        xmm1, QWORD PTR [rbp-40]          divsd   xmm0, xmm1          movsd   QWORD PTR [rbp-48], xmm0          fld     QWORD PTR [rbp-48]          leave          ret  .LC4:          .string "Invalid number of arguments"  .LC5:          .string "N = "  .LC7:          .string "pi = "  .LC8:          .string "Run time: "  .LC9:          .string "\n"  .LC10:          .string "Min time: "  .LC11:          .string " seconds"  main:          push    rbp          mov     rbp, rsp          sub     rsp, 144          mov     DWORD PTR [rbp-132], edi          mov     QWORD PTR [rbp-144], rsi          cmp     DWORD PTR [rbp-132], 2          je      .L9          mov     esi, OFFSET FLAT:.LC4          mov     edi, OFFSET FLAT:std::cerr          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\*)          mov     esi, OFFSET FLAT:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     rdi, rax          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> >&))          mov     eax, 0          jmp     .L13  .L9:          mov     edi, 2          call    sysconf          mov     QWORD PTR [rbp-16], rax          mov     rax, QWORD PTR [rbp-144]          add     rax, 8          mov     rax, QWORD PTR [rax]          mov     rdi, rax          call    atoll          mov     QWORD PTR [rbp-24], rax          mov     esi, OFFSET FLAT:.LC5          mov     edi, OFFSET FLAT:std::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\*)          mov     rdx, rax          mov     rax, QWORD PTR [rbp-24]          mov     rsi, rax          mov     rdi, rdx          call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long long)          mov     esi, OFFSET FLAT:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     rdi, rax          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> >&))          mov     DWORD PTR [rbp-28], 5          movsd   xmm0, QWORD PTR .LC6[rip]          movsd   QWORD PTR [rbp-120], xmm0          mov     QWORD PTR [rbp-8], 0          jmp     .L11  .L12:          lea     rax, [rbp-80]          mov     rdi, rax          call    times          mov     rax, QWORD PTR [rbp-24]          mov     rdi, rax          call    PiCalculation(long long)          fstp    QWORD PTR [rbp-40]          lea     rax, [rbp-112]          mov     rdi, rax          call    times          mov     rdx, QWORD PTR [rbp-112]          mov     rax, QWORD PTR [rbp-80]          sub     rdx, rax          pxor    xmm0, xmm0          cvtsi2sd        xmm0, rdx          pxor    xmm1, xmm1          cvtsi2sd        xmm1, QWORD PTR [rbp-16]          divsd   xmm0, xmm1          movsd   QWORD PTR [rbp-128], xmm0          mov     esi, OFFSET FLAT:.LC7          mov     edi, OFFSET FLAT:std::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\*)          mov     rdx, rax          mov     rax, QWORD PTR [rbp-40]          movq    xmm0, rax          mov     rdi, rdx          call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(double)          mov     esi, OFFSET FLAT:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     rdi, rax          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> >&))          mov     esi, OFFSET FLAT:.LC8          mov     edi, OFFSET FLAT:std::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\*)          mov     rdx, rax          mov     rax, QWORD PTR [rbp-128]          movq    xmm0, rax          mov     rdi, rdx          call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(double)          mov     esi, OFFSET FLAT:.LC9          mov     rdi, rax          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\*)          mov     esi, OFFSET FLAT:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     rdi, rax          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> >&))          lea     rdx, [rbp-128]          lea     rax, [rbp-120]          mov     rsi, rdx          mov     rdi, rax          call    double const& std::min<double>(double const&, double const&)          movsd   xmm0, QWORD PTR [rax]          movsd   QWORD PTR [rbp-120], xmm0          add     QWORD PTR [rbp-8], 1  .L11:          mov     eax, DWORD PTR [rbp-28]          cdqe          cmp     QWORD PTR [rbp-8], rax          jl      .L12          mov     esi, OFFSET FLAT:.LC10          mov     edi, OFFSET FLAT:std::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\*)          mov     rdx, rax          mov     rax, QWORD PTR [rbp-120]          movq    xmm0, rax          mov     rdi, rdx          call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(double)          mov     esi, OFFSET FLAT:.LC11          mov     rdi, rax          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\*)          mov     esi, OFFSET FLAT:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     rdi, rax          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> >&))          mov     eax, 0  .L13:          leave          ret  .LC0:          .long   -4194304          .long   1105199103  .LC1:          .long   0          .long   1072693248  .LC2:          .long   0          .long   1074790400  .LC6:          .long   0          .long   1086556160 |

**Приложение 3: ассемблерный листинг функции *PiCalculation(long long n)* с уровнем -O0, архитектурой x86-64 и комментариями**

|  |
| --- |
| PiCalculation(long long):  push rbp ; cохранение значения регистра rbp (базовый указатель кадра) на стек  mov rbp, rsp ; установка текущего указателя стека (rsp) как базового указателя (rbp)  sub rsp, 48 ; Выделение 48 байт памяти на стеке для локальных переменных  mov QWORD PTR [rbp-40], rdi ; сохранение аргумента функции n (передан через rdi как параметр) в переменную n на стеке по адресу [rbp-40]  mov QWORD PTR [rbp-8], 0 ; инициализация переменной m (счётчик) нулём, хранится по адресу [rbp-8]  mov edi, 0 ; подготовка аргумента для вызова time (в регистр edi передаётся 0)  call time ; вызов функции time, возвращает текущее время в eax  mov edi, eax ; перенос возвращённого значения времени из eax в edi (это аргумент для srand)  call srand ; инициализация генератора случайных чисел функцией srand с переданным значением времени  mov QWORD PTR [rbp-16], 0 ; инициализация счётчика цикла i нулём, хранится по адресу [rbp-16]  jmp .L2 ; переходим в начало цикла  .L5: ; метка цикла (или шаг)  ; **double x = static\_cast<double>(rand()) / RAND\_MAX \* 2.0 - 1.0;**  call rand ; вызов функции rand, возвращает нам случайное число в eax  pxor xmm0, xmm0 ; обнуление регистра xmm0 (для работы с числами с плавающей точкой)  cvtsi2sd xmm0, eax ; преобразование случайного числа (int) из eax в число с плавающей точкой (double), сохраняется в xmm0  movsd xmm1, QWORD PTR .LC0[rip] ; загрузка константы (RAND\_MAX) в xmm1  divsd xmm0, xmm1 ; деление случайного числа на RAND\_MAX (диапазон: [0, 1])  addsd xmm0, xmm0 ; умножение результата на 2 (для диапазона [0, 2])  movsd xmm1, QWORD PTR .LC1[rip] ; загрузка константы 1.0 в xmm1  subsd xmm0, xmm1 ; вычитание 1.0 (чтобы получить искомый диапазон [-1, 1])  movsd QWORD PTR [rbp-24], xmm0 ; Сохранение результата (x) в переменную на стеке ([rbp-24])  **; double y = static\_cast<double>(rand()) / RAND\_MAX \* 2.0 - 1.0;**  call rand ; повтор операций выше для переменной y  pxor xmm0, xmm0  cvtsi2sd xmm0, eax  movsd xmm1, QWORD PTR .LC0[rip]  divsd xmm0, xmml  addsd xmm0, xmm0  movsd xmm1, QWORD PTR .LC1[rip]  subsd xmm0, xmm1  movsd QWORD PTR [rbp-32], xmm0 ; cохранение y в переменную на стеке [rbp-32]  ; **x \* x + y \* y**  movsd xmm0, QWORD PTR [rbp-24] ; загрузка x из стека в xmm0  movapd xmm1, xmm0 ; копирование x в xmm1  mulsd xmm1, xmm0 ; умножение x на x (вычисление x^2)  movsd xmm0, QWORD PTR [rbp-32] ; загрузка y из стека в xmm0  mulsd xmm0, xmm0 ; умножение y на y (вычисление y^2)  addsd xmm1, xmm0 ; сложение x^2 и y^2. Записали в xmm1  **; if (x \* x + y \* y <= 1.0) {**  movsd xmm0, QWORD PTR .LC1[rip] ; загрузка 1.0 в xmm0  comisd xmm0, xmm1 ; сравнение 1.0 с x^2 + y^2  jb .L3 ; переход к .L3, если (x^2 + y^2 > 1) (точка вне круга => не считаем ее)  add QWORD PTR [rbp-8], 1 ; если же точка внутри круга, увеличиваем счётчик m на 1  .L3: **; i++**  add QWORD PTR [rbp-16], 1 ; увеличение счётчика i на 1 (итерация цикла)  .L2:  mov rax, QWORD PTR [rbp-16] ; загружаем текущее значение i из стека в перем rax  cmp rax, QWORD PTR [rbp-40] ; сравниваем i с n  jl .L5 ; если i < n, продолжаем цикл  pxor xmm1, xmm1 ; обнуляем xmm1  cvtsi2sd xmm1, QWORD PTR [rbp-8] ; преобразуем счётчик m в число с плавающей точкой  movsd xmm0, QWORD PTR .LC2[rip] ; загрузка константы 4.0 в регистр xmm0  mulsd xmm0, xmm1 ; умножаем 4.0 на m  pxor xmm1, xmm1 ; обнуляем xmm1  cvtsi2sd xmm1, QWORD PTR [rbp-40] ; преобразуем n в число с плавающей точкой  divsd xmm0, xmm1 ; делим 4.0 \* m на n (**вычисление Pi)**  movsd QWORD PTR [rbp-48], xmm0 ; сохраняем результат в стеке по адресу [rbp-48]  fld QWORD PTR [rbp-48] ; Загружаем результат в FPU стек для возврата  leave ; деаллоцировали локальные, восстановили регистры rbp и rsp  ret ; возврат из функции (по адресу возвата на стеке) |