###### 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, группы 22208

Новикова Григория Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Антон Юрьевич Кудинов

Новосибирск 2023

1. **Цель**

Изучить аспекты работы с языком ассемблера, ознакомиться с архитектурой x86-64 и провести детальный анализ ассемблерного кода программы первой лабораторной работы.

1. **Задание**

Изучить программную архитектуру x86-64. Ознакомиться с набором регистров процессора, основными арифметико-логическими командами, командами арифметического сопроцессора. Изучить способы адресации памяти, работу со стеком, так же понять принцип вызова подпрограмм, передачу параметров в подпрограммы и возврат результатов. Для программы на языке С сгенерировать ассемблерные листинги для архитектуры x86-64, после чего провести анализ ассемблерных листингов, сопоставляя команды языка С с машинными командами. После анализа процессорных инструкций продемонстрировать использование ключевых особенностей архитектуры x86-64 на конкретных участках кода, после чего сравнить различия в программах с оптимизациями и без.

**Листинг программы**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

enum {

REDAING\_ERROR = -1,

WRITING\_ERROR = 1,

CIRCLE\_RADIUS = 1,

WRITING\_SUCCESS = 0

};

double MonteKarlo(const int countPoints) {

srand(time(NULL));

int countPointsInCircle = 0;

for(int i = 0; i < countPoints; ++i) {

double x = rand() / (RAND\_MAX/2.0) - 1.0;

double y = rand() / (RAND\_MAX/2.0) - 1.0;

if (x\*x + y\*y <= CIRCLE\_RADIUS) {

++countPointsInCircle;

}

}

return 4.0\*countPointsInCircle/countPoints;

}

int main() {

const int countPoints = INT\_MAX;

const double pi = MonteKarlo(countPoints);

return EXIT\_SUCCESS;

}

Команда для компиляции:

gcc -O{0, 1, 2, 3, s, g, fast} ./lab3.c -o convert

**Ассемблерный листинг для архитектуры x86-64**

MonteKarlo:

pushq %rbp

movq %rsp, %rbp

subq $48, %rsp // выделяем место под переменные подпрограммы

movl %edi, -36(%rbp)

movl $0, %edi

call time // переходим в подпрограмму time

movl %eax, %edi

call srand

movl $0, -4(%rbp) // инициализируем переменные i и countPointsInCircle

movl $0, -8(%rbp)

jmp .L2 // безусловный переход на метку .L2

.L5:

call rand

pxor %xmm0, %xmm0 // зануляем xmm0

cvtsi2sdl %eax, %xmm0 // конвертируем значение из %eax в значение с плавающей точкой и записываем в векторный регистр

movsd .LC0(%rip), %xmm1 // записываем значение из .LC0 в %xmm1

divsd %xmm1, %xmm0 // делим %xmm0(результат rand()) на %xmm1(RAND\_MAX/2)

movsd .LC1(%rip), %xmm1

subsd %xmm1, %xmm0 // от %xmm0 отнимаем %xmm1(1.0)

movsd %xmm0, -16(%rbp)

call rand

pxor %xmm0, %xmm0

cvtsi2sdl %eax, %xmm0

movsd .LC0(%rip), %xmm1

divsd %xmm1, %xmm0

movsd .LC1(%rip), %xmm1

subsd %xmm1, %xmm0

movsd %xmm0, -24(%rbp)

movsd -16(%rbp), %xmm0

movapd %xmm0, %xmm1

mulsd %xmm0, %xmm1 // умножаем x на x

movsd -24(%rbp), %xmm0

mulsd %xmm0, %xmm0

addsd %xmm0, %xmm1 // сумм x\*x + y\*y

movsd .LC1(%rip), %xmm0

comisd %xmm1, %xmm0 // сравнение x\*x + y\*y с CIRCLE\_RADIUS

jb .L3 // переход если x\*x + y\*y > CIRCLE\_RADIUS

addl $1, -4(%rbp) // увеличиваем countPointsInCircle на 1

.L3:

addl $1, -8(%rbp) // увеличиваем i на 1

.L2:

movl -8(%rbp), %eax

cmpl -36(%rbp), %eax // сравнение i и countPoints

jl .L5 // переход на .L5 если i < countPoints

pxor %xmm1, %xmm1 // вычисляем возвращаемый результат

cvtsi2sdl -4(%rbp), %xmm1

movsd .LC2(%rip), %xmm0

mulsd %xmm1, %xmm0

pxor %xmm1, %xmm1

cvtsi2sdl -36(%rbp), %xmm1

divsd %xmm1, %xmm0

movq %xmm0, %rax

movq %rax, %xmm0

leave // выходим из подпрограммы

ret

main:

pushq %rbp // сохраняем на стеке значение регистра %rbp (base pointer)

movq %rsp, %rbp // записываем значение регистра %rsp (stack pointer) в %rbp, отсекая начало стекового кадра

subq $16, %rsp // отнимаем от %rsp 16 выделяя место на стеке под переменные подпрограммы

movl $2147483647, -4(%rbp) // сохраняем INT\_MAX на стеке

movl -4(%rbp), %eax

movl %eax, %edi

call MonteKarlo // вызываем подпрограмму

movq %xmm0, %rax

movq %rax, -16(%rbp)

movl $0, %eax

leave // выход из программы

ret

.LC0:

.long -4194304

.long 1104150527

.LC1:

.long 0

.long 1072693248

.LC2:

.long 0

.long 1074790400

**Выводы**

По результатам проведённого анализа и сопоставления команд языка С и команд языка ассемблера было выяснено, что в оптимизированном коде значительно уменьшено количество обращений в стек, больше используются регистры общего назначения. Многие вычисления производятся с помощью специальных низкоуровневых операций работающих с битами.