**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 ARM»

студента 2 курса, группы 22208

Новикова Григория Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Антон Юрьевич Кудинов

Новосибирск 2023

**Цель**

Изучить аспекты работы с языком ассемблера; Ознакомиться с архитектурой ARM; Провести детальный анализ ассемблерного кода программы первой лабораторной работы.

**Задание**

Изучить программную архитектуру ARM(Advanced RISC Machine). Ознакомиться с набором регистров процессора, основными отличиями ассемблерного кода архитектуры x86/x86-64 и ARM. Изучить способы адресации памяти, работу со стеком на ARM. Для программы на языке С++ сгенерировать ассемблерный листинг, после чего провести анализ,сопоставляя команды языка С++ с машинными командами. После детального анализа процессорных инструкций продемонстрировать использование ключевых особенностей архитектур ARM на конкретных участках кода.

**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} ./lab4.c -o convert**

**Ассемблерный листинг для O0**

MonteKarlo:

stp x29, x30, [sp, -64]! // снова сохраняем значения x29 и x30 на стек, чтобы отделить место, под локальные переменные одной функции от другой

mov x29, sp // выделяем место под локальные переменные

str w0, [sp, 28]

mov x0, 0

bl time

bl srand

str wzr, [sp, 60] // wzr – регистр содержащий 0, сохраняем на стек тем самым инициализируя переменные I и countPointsInCircle

str wzr, [sp, 56]

b .L2 // переход по метке без сохранения адреса возврата

.L5:

bl rand

scvtf d0, w0 // преобразует значение со знаком в 32-битном или 64- битном исходном регистре общего назначения в значение с плавающей запятой, используя режим округления

mov x0, 281474972516352

movk x0, 0x41cf, lsl 48 // перемещает необязательно сдвинутое 16- битное непосредственное значение в регистр, сохраняя остальные биты неизменными

fmov d1, x0 // перемещаем занчение из x0 в x1

fdiv d1, d0, d1 // делим d0 на d1 и записываем в d1

fmov d0, 1.0e+0 // сохраняем константу 1.0 в d0

fsub d0, d1, d0 // d0 = d1 – d0 – вычислили значение переменной x

str d0, [sp, 48] // сохранили его на стек

bl rand

scvtf d0, w0 // аналогично для y

mov x0, 281474972516352

movk x0, 0x41cf, lsl 48

fmov d1, x0

fdiv d1, d0, d1

fmov d0, 1.0e+0

fsub d0, d1, d0

str d0, [sp, 40]

ldr d0, [sp, 48]

fmul d1, d0, d0 // d1 = x \* x

ldr d0, [sp, 40]

fmul d0, d0, d0 // d0 = y \* y

fadd d1, d1, d0 // d1 = x\*x + y\*y

fmov d0, 1.0e+0

fcmpe d1, d0 // сравниваем x\*x + y\*y с CIRCLE\_RADIUS(=1.0)

bls .L7 // переход по метке если x\*x + y\*y <= CIRCLE\_RADIUS

b .L3

.L7:

ldr w0, [sp, 60]

add w0, w0, 1 // увеличиваем countPointsInCircle на 1

str w0, [sp, 60]

.L3:

ldr w0, [sp, 56]

add w0, w0, 1 // увеличиваем i на 1

str w0, [sp, 56]

.L2:

ldr w1, [sp, 56] // сохраняем значение переменной i со стека в регистр

ldr w0, [sp, 28] // аналогично для переменной countPoints

cmp w1, w0 // сравниваем значения регистров и устанавливаем флаги

blt .L5 // переход по метке если i < countPoints

ldr w0, [sp, 60] // вычисляем возвращаемое значение

scvtf d1, w0

fmov d0, 4.0e+0

fmul d1, d1, d0

ldr w0, [sp, 28]

scvtf d0, w0

fdiv d0, d1, d0

ldp x29, x30, [sp], 64

ret

main:

stp x29, x30, [sp, -32]! // сохраняем значения регистров x29(frame pointer и x30(link register) на стек

mov x29, sp // сохраняем занчение регистра sp(stack pointer) в регистр x29

mov w0, 2147483647 // записываем INT\_MAX в w0

str w0, [sp, 28] // записываем значение регистра w0 в стек

ldr w0, [sp, 28]

bl MonteKarlo // переходим по метке MonteKarlo и сохраняем адрес возврата в регистре X30

str d0, [sp, 16]

mov w0, 0 // обнуляем регистр w0

ldp x29, x30, [sp], 32 // считываем старые значения x29 и x30 со стека

ret // выход из программы

**Выводы**

По результатам проведенного анализа было установлено, что в ассемблерном коде ARM обращение к памяти, работа с математическими функциями, работа с регистрами существенно отличаются от ассемблерного кода на x86/x86-64. Изучены основные отличия этих архитектур и изучены основы ARM.