###### 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, 18209 группы

**Большим Максима Антоновича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2019

1. **Цель**

1. Изучить аспекты работы с языком ассемблера;

2. Ознакомиться с архитектурой ARM;

3. Провести детальный анализ ассемблерного кода программы 1й лабораторной работы.

1. **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.

1. **Листинг программы**

**#include <iostream>**

**inline double power(unsigned long power)**

**{**

**if(power % 2 == 1)**

**return -1.0;**

**else return 1.0;**

**}**

**int main()**

**{**

**unsigned long N = 4200000000;**

**double Pi = 0;**

**for(unsigned long i = 0;i < N;i++)**

**Pi += power(i) \* (4.0L /**

**(2.0L \* static\_cast<long double>(i) + 1.0L));**

**printf("%.20f\n", Pi);**

**}**

1. **Ассемблерный листинг для архитектуры x86-64**

\_\_gthread\_key\_delete(int):

str fp, [sp, #-4]!

add fp, sp, #0

sub sp, sp, #12

str r0, [fp, #-8]

mov r3, #0

mov r0, r3

add sp, fp, #0

ldr fp, [sp], #4

bx lr

std::\_\_ioinit:

power(unsigned long):

push {r4, fp} //save registers

add fp, sp, #4 //work with stack pointer

sub sp, sp, #8

str r0, [fp, #-8] //save r0=i

ldr r3, [fp, #-8] //r0 to r3, r3=i

and r3, r3, #1 //check i mod 2

cmp r3, #0 //compare i & 1 == 0

beq .L4 //r3 = 0 and r4 = 1.0

mov r3, #0 //else r4=-1.0

ldr r4, .L6 //

b .L5 //jmp to L5

.L4:

mov r3, #0

ldr r4, .L6+4

.L5:

mov r0, r3 //some manupulations

mov r1, r4 //r1 = 1.0 or -1.0

sub sp, fp, #4 //restore old registers

pop {r4, fp}

bx lr //return to main

.L6:

.word -1074790400

.word 1072693248

.LC0:

.ascii "%.20f\012\000"

main:

push {r4, r5, fp, lr} //save registers

add fp, sp, #12 //fp = sp + 12 for saved registers

sub sp, sp, #16 //new var in stack

ldr r3, .L12 //N = 4200000000

str r3, [fp, #-28] //store N

mov r3, #0 //Pi = 0.0

mov r4, #0 //

str r3, [fp, #-20] //

str r4, [fp, #-16] //

mov r3, #0 //i = 0

str r3, [fp, #-24] //store i

.L10:

ldr r2, [fp, #-24] //load i

ldr r3, [fp, #-28] //load N

cmp r2, r3 //i comp with N

bcs .L9 //i >= N

ldr r0, [fp, #-24] //r0 = i for power()

bl power(unsigned long)

mov r4, r0

mov r5, r1

ldr r0, [fp, #-24] //r0=i

bl \_\_aeabi\_ui2d //i to double r0=i

mov r2, r0

mov r3, r1

bl \_\_aeabi\_dadd //i\*2

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3

mov r2, #0

ldr r3, .L12+4 //load 1.0

bl \_\_aeabi\_dadd //2i+1

mov r2, r0

mov r3, r1

mov r0, #0

ldr r1, .L12+8 //load 4.0

bl \_\_aeabi\_ddiv //4.0 / (2i+1)

mov r2, r0

mov r3, r1

mov r0, r4

mov r1, r5

bl \_\_aeabi\_dmul //+-11.0 \* 4.0 / (2i+1)

mov r3, r0

mov r4, r1

mov r2, r3

mov r3, r4

sub r1, fp, #20

ldmia r1, {r0-r1}

bl \_\_aeabi\_dadd //load Pi and Pi += (+-11.0 \* 4.0 / (2i+1))

mov r3, r0

mov r4, r1

str r3, [fp, #-20]

str r4, [fp, #-16] //store Pi

ldr r3, [fp, #-24] //load i and i++ and store i

add r3, r3, #1

str r3, [fp, #-24]

b .L10

.L9:

sub r3, fp, #20 //take base point

ldmia r3, {r2-r3} //load Pi

ldr r0, .L12+12 //load format

bl printf

mov r3, #0

mov r0, r3 //r0=r3=0

sub sp, fp, #12 //restore data

pop {r4, r5, fp, lr} //restore old values of registers

bx lr //return 0;

.L12:

.word -94967296

.word 1072693248

.word 1074790400

.word .LC0

\_\_static\_initialization\_and\_destruction\_0(int, int): //init iostream

push {fp, lr}

add fp, sp, #4

sub sp, sp, #8

str r0, [fp, #-8]

str r1, [fp, #-12]

ldr r3, [fp, #-8]

cmp r3, #1

bne .L16

ldr r3, [fp, #-12]

ldr r2, .L17

cmp r3, r2

bne .L16

ldr r0, .L17+4

bl std::ios\_base::Init::Init() [complete object constructor]

ldr r2, .L17+8

ldr r1, .L17+12

ldr r0, .L17+4

bl \_\_aeabi\_atexit

.L16:

nop

sub sp, fp, #4

pop {fp, lr}

bx lr

.L17:

.word 65535

.word std::\_\_ioinit

.word \_\_dso\_handle

.word \_ZNSt8ios\_base4InitD1Ev

\_GLOBAL\_\_sub\_I\_main: //GLOBAL main

push {fp, lr}

add fp, sp, #4

ldr r1, .L20

mov r0, #1

bl \_\_static\_initialization\_and\_destruction\_0(int, int)

sub sp, fp, #4

pop {fp, lr}

bx lr

.L20:

.word 65535

1. **Выводы**

По результатам проведённого 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. Так же изучены основные отличия этих архитектур и изучены основы ARM.