###### 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 (ки) 2 курса, 18205 группы

Гайдамака Андрея Владиславовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

(ученая степень, звание)

А. Ю. Власенко

Новосибирск 2019

ЦЕЛЬ 3

ЗАДАНИЕ 3

ОПИСАНИЕ РАБОТЫ 3

ЗАКЛЮЧЕНИЕ 4

Приложение 1. Код программы epower 5

Приложение 2. Ассемблерный листинг……………………………………7

# ЦЕЛЬ

1. Знакомство с программной архитектурой ARM.
2. Анализ ассемблерного листинга программы для архитектуры ARM.

# ЗАДАНИЕ

1. Изучить программную архитектуру ARM
2. Для программы на языке Си (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры ARM
3. Проанализировать полученные листинги.

**Вариант задания: №5**

# ОПИСАНИЕ РАБОТЫ

После изучения программной архитектуры ARM, был проанализирован ассемблерный листинг (Приложение 2) программы на языке Си (из лабораторной работы №1)(Приложение 1). Во время разбора были сопоставлены инструкции на языке Си с машинными программами. Так же было определено размещение переменных, регистры в которых передаются аргументы в функцию и возвращаются из нее.

# ЗАКЛЮЧЕНИЕ

Можно сделать вывод, что рассмотрение ассемблерного листинга(Приложение 2) программы помогает в разборе работы функций, циклов, условий на очень низком уровне, что сильно помогает в оптимизации программы. Так же была сравнена простота кода risc архитектуры по сравнению с cisc.

# Приложение 1. Код программы epower

*#include <stdio.h>*

*#include <stdlib.h>*

*int factorial(const int n){*

*int res = 1;*

*for (int i = 1; i <= n; ++i){*

*res \*= i;*

*}*

*return res;*

*}*

*int power(int a, int n){*

*int res = 1;*

*for (int i = 0; i < n; ++i){*

*res \*= a;*

*}*

*return res;*

*}*

*double epower(const long long int n, const long long int x){*

*double epow = 0;*

*double last = 0;*

*double tmp = 0;*

*for(long int i = 0; i <= n; ++i){*

*if (i <= 7){*

*last = (double)power(x, i) / factorial(i);*

*epow += last;*

*}*

*else{*

*last \*= (double)((double)x / i);*

*epow += last;*

*}*

*}*

*return epow;*

*}*

*int main(int argc, char \*\* argv) {*

*if (argc != 3){*

*return 1;*

*}*

*long long int n = atoll(argv[1]);*

*long long int x = atoll(argv[2]);*

*double epow = epower(n, x);*

*printf("%f\n", epow);*

*return 0;*

*}*

**Приложение 2. Ассемблерный Листинг**

\_Z9factoriali:

str  fp, [sp, #-4]!

add  fp, sp, #0

sub  sp, sp, #20

str  r0, [fp, #-16] //n

mov  r3, #1

str  r3, [fp, #-8] //res

mov  r3, #1

str  r3, [fp, #-12] //i

.L3:

ldr  r2, [fp, #-12] //цикл for

ldr  r3, [fp, #-16]

cmp  r2, r3 //сравнение n и i

bgt  .L2

ldr  r3, [fp, #-8]

ldr  r2, [fp, #-12]

mul  r1, r2, r3

str  r1, [fp, #-8]

ldr  r3, [fp, #-12]

add  r3, r3, #1

str  r3, [fp, #-12]

b  .L3

.L2:

ldr  r3, [fp, #-8]

mov  r0, r3

add  sp, fp, #0

ldr  fp, [sp], #4

bx  lr

\_Z5powerii:

str  fp, [sp, #-4]!

add  fp, sp, #0

sub  sp, sp, #20

str  r0, [fp, #-16] //a

str  r1, [fp, #-20] //n

mov  r3, #1

str  r3, [fp, #-8] //res

mov  r3, #0

str  r3, [fp, #-12] //i

.L7:

ldr  r2, [fp, #-12] //цикл for

ldr  r3, [fp, #-20]

cmp  r2, r3

bge  .L6

ldr  r3, [fp, #-8]

ldr  r2, [fp, #-16]

mul  r1, r2, r3

str  r1, [fp, #-8]

ldr  r3, [fp, #-12]

add  r3, r3, #1

str  r3, [fp, #-12]

b  .L7

.L6:

ldr  r3, [fp, #-8]

mov  r0, r3

add  sp, fp, #0

ldr  fp, [sp], #4

bx  lr

\_Z6epowerxx:

push {r4, r5, fp, lr}

add  fp, sp, #12

sub  sp, sp, #48

str  r0, [fp, #-52] //n

str  r1, [fp, #-48] //n

str  r2, [fp, #-60] //x

str  r3, [fp, #-56] //x

mov  r3, #0

mov  r4, #0

str  r3, [fp, #-20] //epow

str  r4, [fp, #-16] //epow

mov  r3, #0

mov  r4, #0

str  r3, [fp, #-28] //last

str  r4, [fp, #-24] //last

mov  r3, #0

mov  r4, #0

str  r3, [fp, #-44] //tmp

str  r4, [fp, #-40] //tmp

mov  r3, #0

str  r3, [fp, #-32] //i

.L13:

ldr  r3, [fp, #-32]

asr  r4, r3, #31 //arifmetic shift right

sub  r2, fp, #52

ldmia  r2, {r1-r2}

cmp  r1, r3

sbcs  r3, r2, r4. //обратное вычитание

blt  .L10

ldr  r3, [fp, #-32]

cmp  r3, #7

bgt  .L11

ldr  r3, [fp, #-60]

ldr  r1, [fp, #-32]

mov  r0, r3

bl  \_Z5powerii

mov  r3, r0

mov  r0, r3

bl  \_\_aeabi\_i2d

mov  r4, r0

mov  r5, r1

ldr  r0, [fp, #-32]

bl  \_Z9factoriali

mov  r3, r0

mov  r0, r3

bl  \_\_aeabi\_i2d //конвертация int в double

mov  r2, r0

mov  r3, r1

mov  r0, r4

mov  r1, r5

bl  \_\_aeabi\_ddiv // деление double

mov  r3, r0

mov  r4, r1

str  r3, [fp, #-28]

str  r4, [fp, #-24]

sub  r3, fp, #28

ldmia  r3, {r2-r3}

sub  r1, fp, #20

ldmia  r1, {r0-r1}

bl  \_\_aeabi\_dadd //сложение double

mov  r3, r0

mov  r4, r1

str  r3, [fp, #-20]

str  r4, [fp, #-16]

b  .L12

.L11:

sub  r1, fp, #60

ldmia  r1, {r0-r1}

bl  \_\_aeabi\_l2d //конвертация long в double

mov  r4, r0

mov  r5, r1

ldr  r0, [fp, #-32]

bl  \_\_aeabi\_i2d //конвертация int в double

mov  r2, r0

mov  r3, r1

mov  r0, r4

mov  r1, r5

bl  \_\_aeabi\_ddiv //деление double

mov  r3, r0

mov  r4, r1

mov  r2, r3

mov  r3, r4

sub  r1, fp, #28

ldmia  r1, {r0-r1}

bl  \_\_aeabi\_dmul //умножение double

mov  r3, r0

mov  r4, r1

str  r3, [fp, #-28]

str  r4, [fp, #-24]

sub  r3, fp, #28

ldmia  r3, {r2-r3}

sub  r1, fp, #20

ldmia  r1, {r0-r1}

bl  \_\_aeabi\_dadd //сложение double

mov  r3, r0

mov  r4, r1

str  r3, [fp, #-20]

str  r4, [fp, #-16]

.L12:

ldr  r3, [fp, #-32]

add  r3, r3, #1

str  r3, [fp, #-32]

b  .L13

.L10:

sub  r4, fp, #20

ldmia  r4, {r3-r4}

mov  r0, r3

mov  r1, r4

sub  sp, fp, #12

pop {r4, r5, fp, pc}

main:

push {fp, lr}

add  fp, sp, #4

sub  sp, sp, #32

str  r0, [fp, #-32] //argc

str  r1, [fp, #-36] //argv

ldr  r3, [fp, #-32]

cmp  r3, #3

beq  .L16

mov  r3, #1

b  .L17

.L16:

ldr  r3, [fp, #-36]

add  r3, r3, #4 //argv[1]

ldr  r3, [r3] //\*argv[1]

mov  r0, r3 //atoll принимает аргумент в регистре r0

bl  atoll

str  r0, [fp, #-12] //n

str  r1, [fp, #-8] //n

ldr  r3, [fp, #-36]

add  r3, r3, #8

ldr  r3, [r3]

mov  r0, r3

bl  atoll

str  r0, [fp, #-20] //x

str  r1, [fp, #-16] //x

sub  r3, fp, #20

ldmia  r3, {r2-r3}

sub  r1, fp, #12

ldmia  r1, {r0-r1}

bl  \_Z6epowerxx

str  r0, [fp, #-28] //epow

str  r1, [fp, #-24] //epow

mov  r3, #0

.L17:

mov  r0, r3

sub  sp, fp, #4

pop {fp, pc}

*0t*