**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 19201

Хаецкой Дарьи Владимировны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2020

**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 <stdlib.h>**

**#include <math.h>**

**#include <time.h>**

**#define pi 3.14159265**

**#define N 200000000**

**long double sinus(double x){**

**long double current, prev = x, sin = 0;**

**long double numenator = 0, denumenator = 0;**

**for (int n = 1, f = 1; n < N; n++, f+=2){**

**sin += prev;**

**prev = abs(prev);**

**numenator = pow(-1, n) \* pow(x,2);**

**denumenator = (f + 1) \* (f + 2);**

**current = prev \* numenator / denumenator;**

**prev = current;**

**}**

**return sin;**

**}**

**double toRadians(double degrees){**

**return (pi / 180) \* degrees;**

**}**

**int main() {**

**struct timespec start, end;**

**int degrees = 90;**

**clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);**

**sinus(toRadians(degrees));**

**clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);**

**double time = end.tv\_sec-start.tv\_sec + 1e-9 \* (end.tv\_nsec - start.tv\_nsec);**

**printf("%ld \n", time);**

**return 0;**

**}**

**Ассемблерный листинг для O0**

sinus:

stp x29, x30, [sp, -144]! // Store Pair Registers (extended): stores two doublewords from X29 and X30 to memory addressed

mov x29, sp // move sp to Frame Pointer

str d8, [sp, 16] // Store double Register. 64-bit doubleword registers, D0-D31.

str d0, [sp, 40] // Load Register: loads a word from memory addressed by addr to Wt.

ldr d0, [sp, 40] // загружаем х в prev

bl \_\_extenddftf2 // Branch and Link: unconditionally jumps to pc-relative

// label writing the address of the next sequential instruction to register X30

// это было приведение double к long double

str q0, [sp, 128] // load 128-bit quadword register q0

stp xzr, xzr, [sp, 112] // xzr - Zero register sin = 0

stp xzr, xzr, [sp, 80] // numenator = 0

stp xzr, xzr, [sp, 64] // denumenator = 0

mov w0, 1 // int n = 1

str w0, [sp, 108]

mov w0, 1 // int f = 1

str w0, [sp, 104]

b .L2 // Branch: unconditionally jumps to pc-relative label

.L3:

ldr q1, [sp, 128]

ldr q0, [sp, 112]

bl \_\_addtf3 // function returns the sum of a and b.

str q0, [sp, 112] // в q0 загружается значение суммы sin

ldr q0, [sp, 128] // загрузить эту сумму в sin

// округление

bl \_\_fixtfsi // functions convert a to a signed integer, rounding toward zero

cmp w0, 0

csneg w0, w0, w0, ge // Conditional select negation, returning the first input or negated second input

bl \_\_floatsitf

str q0, [sp, 128] // сохранение результата в prev

// numenator = pow(-1, n) \* pow(x,2);

ldr w0, [sp, 108] // загружаем n в регистр

scvtf d0, w0 // converts the signed value in the 32-bit or 64-bit general-purpose source register to a floating-point value using the rounding mode

fmov d1, d0

fmov d0, -1.0e+0 // Move 64 bits unchanged записали -1

bl pow

fmov d8, d0 // Move 64 bits unchanged form d0 to d8

fmov d1, 2.0e+0 // записали -2

ldr d0, [sp, 40] // загружаем х в регистр

bl pow

fmul d0, d8, d0 // перемножаем и пишем в d0

bl \_\_extenddftf2 // приведение к long double

str q0, [sp, 80]

// denumenator = (f + 1) \* (f + 2);

ldr w0, [sp, 104] // загрузить f

add w1, w0, 1 // w1 = f + 1

ldr w0, [sp, 104] // загрузить f снова

add w0, w0, 2 // w0 = f + 2

mul w0, w1, w0 // w0 = (f + 1) \* (f + 2)

bl \_\_floatsitf // These functions convert i, a signed integer, to floating point

str q0, [sp, 64] //

//current = prev \* numenator / denumenator;

ldr q1, [sp, 80]

ldr q0, [sp, 128]

bl \_\_multf3 // return the product of q1 and q0

ldr q1, [sp, 64]

bl \_\_divtf3 // q0 = q0 / q1

str q0, [sp, 48]

ldr q0, [sp, 48] // prev = current

str q0, [sp, 128]

ldr w0, [sp, 108]

add w0, w0, 1 // n += 1

str w0, [sp, 108]

ldr w0, [sp, 104]

add w0, w0, 2 // f += 2

str w0, [sp, 104]

.L2:

ldr w1, [sp, 108] // загружаем n в w1

mov w0, 49663 // записываем N в w0

movk w0, 0xbeb, lsl 16 // w0 |= (0xbeb << 16)

cmp w1, w0 // сравниваем n и N

ble .L3 // переход если меньше либо равно

ldr q0, [sp, 112] // выгружаем sin

ldr d8, [sp, 16] // и х

ldp x29, x30, [sp], 144 // x30 is the link register and x29 is the frame pointer

// link register is a special-purpose register which holds

// the address to return to when a function call completes.

ret

toRadians:

sub sp, sp, #16 // сдвинули указатель на 16

str d0, [sp, 8] // место для degrees

ldr d0, [sp, 8] // загружаем degrees

adrp x0, .LC0 //

ldr d1, [x0, #:lo12:.LC0] // загрузка числа пи / 180

fmul d0, d0, d1 // (pi / 180) \* degrees

add sp, sp, 16 // сдвиг указателя стека?

ret

.LC1:

.string "%ld \n"

main:

stp x29, x30, [sp, -64]! // Store Pair Registers (extended): stores two doublewords from X29 and X30 to memory addressed

mov x29, sp // move sp to Frame Pointer

mov w0, 90 // int degrees = 90;

str w0, [sp, 60]

add x0, sp, 32 // x0 = [sp, 32]

mov x1, x0 // x1 = x0

mov w0, 4 // w0 = 4

bl clock\_gettime

ldr w0, [sp, 60] // запись результата в start

scvtf d0, w0 // converts the signed value in the 32-bit or 64-bit

// general-purpose source register to a floating-point value using the rounding mode

bl toRadians

bl sinus

add x0, sp, 16 // x0 = [sp, 16]

mov x1, x0

mov w0, 4

bl clock\_gettime

ldr x1, [sp, 16] // загрузить end.tv\_nsec

ldr x0, [sp, 32] // загрузить start.tv\_nsec

sub x0, x1, x0 // end.tv\_nsec - start.tv\_nsec

scvtf d1, x0 // converts the signed value in the 32-bit or 64-bit

// general-purpose source register to a floating-point value using the rounding mode

ldr x1, [sp, 24] // загрузить end.tv\_sec

ldr x0, [sp, 40] // загрузить start.tv\_sec

sub x0, x1, x0 // end.tv\_sec - start.tv\_sec

scvtf d0, x0 // приведение к double

adrp x0, .LC2 // Address of 4KB page at a PC-relative offset.

// LC2 is the program label whose 4KB page address is to be calculated.

// shifts pages (4KiB, P in ADRP stands for Page) relative to the

// current instead of just bytes zeroes out the 12 lower bits

ldr d2, [x0, #:lo12:.LC2] // Since the lower 12 bits are zeroed out, to calculate a full address

// we use ADD + :lo12:

// тут в d2 загружается константа 1e-9

fmul d0, d0, d2 // на неё умножается ранее рассчитанное значение

fadd d0, d1, d0 // всё складывается

str d0, [sp, 48] // и записывается в time

ldr d0, [sp, 48] // выгружаем time в регистр

adrp x0, .LC1 // всё как выше

add x0, x0, :lo12:.LC1

bl printf

mov w0, 0 // return 0

ldp x29, x30, [sp], 64

.LC0:

.word 2717574930

.word 1066524486

.LC2:

.word 3894859413

.word 1041313291

**Ассемблерный листинг для архитектуры ARM с оптимизацией O2**

sinus:

movi v4.2d, #0 // sin = 0

stp x29, x30, [sp, -96]! // Store Pair Registers (extended): stores two doublewords from X29 and X30 to memory addressed

mov x29, stp // move sp to Frame Pointer

str d8, [sp, 56] // Store double Register. 64-bit doubleword registers, D0-D31.

fmov d8, d0 // записали х в d8

stp x19, x20, [sp, 16] //

mov w20, 1 // n = 1

stp x21, x22, [sp, 32]

mov w21, w20 // f = 1

str x23, [sp, 48] //

mov w23, 49664 // w23 = 1001001011001100100

str q4, [sp, 64] // load 128-bit quadword register q4

bl \_\_extenddftf2 // приведение х к long double

fmul d8, d8, d8 // х заранее возводится в квадрат, оптимизация компилятора

mov v2.16b, v0.16b // копируем первые восемь бит x в prev

ldr q4, [sp, 64] //

movk w23, 0xbeb, lsl 16 // w23 |= (0xbeb << 16) получаем N

.L2:

mov v0.16b, v4.16b // в v0 пишется sin

mov v1.16b, v2.16b // в v1 пишется prev

str q2, [sp, 64] // память под prev

bl \_\_addtf3 // складываем sin и prev

ldr q2, [sp, 64] // память под sin

add w22, w20, 1

str q0, [sp, 80]

add w20, w20, 2 // f += 2

mov v0.16b, v2.16b

bl \_\_fixtfsi // functions convert a to a signed integer, rounding toward zero

scvtf d1, w21 // converts the signed value in the 32-bit or 64-bit general-purpose source register to a floating-point value using the rounding mode

cmp w0, 0

fmov d0, -1.0e+0 // d0 = -1

csneg w19, w0, w0, ge // Conditional select negation, returning the first input or negated second input

add w21, w21, 1 //

bl pow // pow (-1, n)

fmul d0, d8, d0 // pow (-1, n) \* x\*\*2

bl \_\_extenddftf2 // привести это в long double

str q0, [sp, 64]

mov w0, w19

bl \_\_floatsitf // abs(prev)

ldr q2, [sp, 64]

mov v1.16b, v0.16b // v1 = numenator

mov v0.16b, v2.16b // v0 = prev

bl \_\_multf3 // prev \* numenator

mul w0, w20, w22 // (f + 1) \* (f + 2)

str q0, [sp, 64]

bl \_\_floatsitf

mov v1.16b, v0.16b

ldr q2, [sp, 64]

mov v0.16b, v2.16b

bl \_\_divtf3 // prev \* numenator / denumenator

cmp w21, w23 // n < N

mov v2.16b, v0.16b

ldr q4, [sp, 80]

bne .L2 // если меньше то снова идём вверх

mov v0.16b, v4.16b

ldr d8, [sp, 56]

ldp x19, x20, [sp, 16] // Load Pair Registers (extended): loads two doublewords from memory

ldp x21, x22, [sp, 32]

ldr x23, [sp, 48]

ldp x29, x30, [sp], 96

ret

toRadians:

adrp x0, .LC0

ldr d1, [x0, #:lo12:.LC0] // загружаем (pi / 180)

fmul d0, d0, d1 // (pi / 180) \* degrees

ret

main:

stp x29, x30, [sp, -48]! // move sp to Frame Pointer

mov w0, 4

mov x29, sp // move sp to Frame Pointer

add x1, sp, 16 // x1 = [sp, 16]

bl clock\_gettime

adrp x0, .LC2

ldr d0, [x0, #:lo12:.LC2]

bl sinus

add x1, sp, 32 // x1 = [sp, 32]

mov w0, 4 //

bl clock\_gettime

ldp x2, x1, [sp, 16] // загрузить end.tv\_sec и end.tv\_nsec

adrp x0, .LC3

ldp d1, d0, [sp, 32] // загрузить start.tv\_sec и start.tv\_nsec

ldr d2, [x0, #:lo12:.LC3] // загружаем 1e-9

adrp x0, .LC1

add x0, x0, :lo12:.LC1 // получаем аргуметы printf

fmov d3, x2

sub d1, d1, d3 // end.tv\_nsec - start.tv\_nsec

fmov d3, x1

sub d0, d0, d3 // end.tv\_sec-start.tv\_sec

scvtf d1, d1

scvtf d0, d0

fmadd d0, d0, d2, d1 // Double-precision floating-point scalar fused multiply-add: Dd = Da + Dn\*Dm.

bl printf

mov w0, 0 // return 0

ldp x29, x30, [sp], 48

ret

.LC0:

.word 2717574930

.word 1066524486

.LC2:

.word 1405670641

.word 1073291771

.LC3:

.word 3894859413

.word 1041313291

.LC1:

.string "%ld \n"

**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.