###### **МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

###### **РОССИЙСКОЙ ФЕДЕРАЦИИ**

###### **НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

###### 

###### 

**ЭВМ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

ВВЕДЕНИЕ В АРХИТЕКТУРУ ARM

Студент:

Овчаренко Дарья Ивановна, группа 23211

Преподаватель:

Ассистент кафедры ПВ ФИТ

Мичуров Михаил Антонович

Новосибирск 2024

# **ЦЕЛЬ**

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

# **ЗАДАНИЕ**

Вариант задания: 4.

1. Изучить программную архитектуру ARM: набор регистров, основные арифметико-логические команды, способы адресации памяти, способы передачи управления, работу со стеком, вызов подпрограмм, передачу параметров в подпрограммы и возврат результатов, работу с арифметическим сопроцессором, работу с векторными расширениями.
2. Сгенерировать листинги исходной программы с оптимизациями –O0, -O1, -Ofast и проанализировать полученные коды.
3. Составить отчет по лабораторной работе.

# **ОПИСАНИЕ РАБОТЫ**

В ходе задания использовался компьютер с архитектурой x86\_64, с операционной системой Ubuntu 22.04.5 LTS и процессором AMD A6-6310 APU with AMD Radeon R4 Graphics.

###### **Пошаговое описание выполненной работы**

1. Была написана компьютерная программа, которая вычисляет sin(x) с помощью разложения в степенной ряд по первым N членам этого ряда.
2. Были изучены основные принципы работы в языке ассемблер.
3. С помощью сайта GodBolt (URL: https://godbolt.org/) были сгенерированы листинги исходной программы с оптимизациями -O0 (см. Приложение 2), -O1 (см. Приложение 3) и -Ofast (см. Приложение 3).

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения не только лабораторной работы №4, но и также предыдущей лабораторной работы №3, было осуществлено знакомство с архитектурами CISC и RISC и сделаны некоторые выводы об их различиях. Различия также были рассмотрены между ассемблерными листингами, сгенерированными под разные архитектуры.

В архитектуре ARM используются другие команды и регистры. ARM использует простую, компактную и энергоэффективную архитектуру RISC (Reduced Instruction Set Computing), которая позволяет выполнять команды быстрее и с меньшими затратами энергии по сравнению с CISC (Complex Instruction Set Computing) архитектурами, такими как x86. Команды ARM оптимизированы для выполнения за один такт процессора. Такой подход позволяет сократить задержки и снизить энергопотребление, что особенно полезно для мобильных устройств.

# **ПРИЛОЖЕНИЕ 1.** Листинг программы с библиотечной функцией clock\_gettime на языке Си

#define \_POSIX\_C\_SOURCE 199309L

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#define PI 3.1415926535897

double CalcSin(double x, long long n);

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

if (argc < 3) {

printf("Bad input! Few arguments. Enter x and n in command line.");

return -1;

}

if (argc > 3){

printf("Bad input! A lot of arguments. Enter x and n in command line.");

return -1;

}

struct timespec res, start, end;

if (clock\_getres(CLOCK\_MONOTONIC\_RAW, &res) == 0){ // сравнивается с 0, так как 0 значит успешное выполнение

printf("Timer resolution: %ld sec, %ld nanosec.\n", res.tv\_sec, res.tv\_nsec);

} else {

perror("Call error clock\_getres!");

}

char \*endptr\_x, \*endptr\_n;

long x = strtol(argv[1], &endptr\_x, 10);

long long n = strtol(argv[2], &endptr\_n, 10);

if (\*endptr\_x != '\0'){

printf("Error: Invalid input for x: %s\n", argv[1]);

return -1;

}

if (\*endptr\_n != '\0'){

printf("Error: Invalid input for n: %s\n", argv[2]);

return -1;

}

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

double sinx = CalcSin((double)x, n);

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

printf("%lf\n", sinx);

printf("Time takken: %lf sec.\n", end.tv\_sec - start.tv\_sec + 0.000000001\*(end.tv\_nsec-start.tv\_nsec));

return 0;

}

double CalcSin(double x, long long n){

double sinx = 0;

x = x \* PI / 180; // перевод градусы в радианы, иначе некорректные вычисления

double sum = x;

for (long long i = 1; i <= 2 \* n - 1; i += 2){

sinx += sum;

sum = (sum \* x \* x \* (-1)) / ((i + 1) \* (i + 2));

}

return sinx;

}

**ПРИЛОЖЕНИЕ 2.** Ассемблерный листинг кода с оптимизацией -O0 (ARM GCC 14.2.0)

.LC0:

.ascii "Bad input! Few arguments. Enter x and n in command "

.ascii "line.\000"

.LC1:

.ascii "Bad input! A lot of arguments. Enter x and n in com"

.ascii "mand line.\000"

.LC2:

.ascii "Timer resolution: %ld sec, %ld nanosec.\012\000"

.LC3:

.ascii "Call error clock\_getres!\000"

.LC4:

.ascii "Error: Invalid input for x: %s\012\000"

.LC5:

.ascii "Error: Invalid input for n: %s\012\000"

.LC6:

.ascii "%lf\012\000"

.LC7:

.ascii "Time takken: %lf sec.\012\000"

main:

push {r4, r5, r7, lr}

sub sp, sp, #64

add r7, sp, #0

str r0, [r7, #4]

str r1, [r7]

ldr r3, [r7, #4]

cmp r3, #2

bgt .L2

movw r0, #:lower16:.LC0

movt r0, #:upper16:.LC0

bl printf

mov r3, #-1

b .L9

.L2:

ldr r3, [r7, #4]

cmp r3, #3

ble .L4

movw r0, #:lower16:.LC1

movt r0, #:upper16:.LC1

bl printf

mov r3, #-1

b .L9

.L4:

add r3, r7, #32

mov r1, r3

movs r0, #4

bl clock\_getres

mov r3, r0

cmp r3, #0

bne .L5

ldr r3, [r7, #32]

ldr r2, [r7, #36]

mov r1, r3

movw r0, #:lower16:.LC2

movt r0, #:upper16:.LC2

bl printf

b .L6

.L5:

movw r0, #:lower16:.LC3

movt r0, #:upper16:.LC3

bl perror

.L6:

ldr r3, [r7]

adds r3, r3, #4

ldr r3, [r3]

add r1, r7, #12

movs r2, #10

mov r0, r3

bl strtol

str r0, [r7, #60]

ldr r3, [r7]

adds r3, r3, #8

ldr r3, [r3]

add r1, r7, #8

movs r2, #10

mov r0, r3

bl strtol

mov r3, r0

asrs r2, r3, #31

mov r4, r3

mov r5, r2

strd r4, [r7, #48]

ldr r3, [r7, #12]

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

beq .L7

ldr r3, [r7]

adds r3, r3, #4

ldr r3, [r3]

mov r1, r3

movw r0, #:lower16:.LC4

movt r0, #:upper16:.LC4

bl printf

mov r3, #-1

b .L9

.L7:

ldr r3, [r7, #8]

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

beq .L8

ldr r3, [r7]

adds r3, r3, #8

ldr r3, [r3]

mov r1, r3

movw r0, #:lower16:.LC5

movt r0, #:upper16:.LC5

bl printf

mov r3, #-1

b .L9

.L8:

add r3, r7, #24

mov r1, r3

movs r0, #4

bl clock\_gettime

ldr r3, [r7, #60]

vmov s15, r3 @ int

vcvt.f64.s32 d16, s15

ldrd r0, [r7, #48]

vmov.f64 d0, d16

bl CalcSin

vstr.64 d0, [r7, #40]

add r3, r7, #16

mov r1, r3

movs r0, #4

bl clock\_gettime

ldrd r2, [r7, #40]

movw r0, #:lower16:.LC6

movt r0, #:upper16:.LC6

bl printf

ldr r2, [r7, #16]

ldr r3, [r7, #24]

subs r3, r2, r3

vmov s15, r3 @ int

vcvt.f64.s32 d17, s15

ldr r2, [r7, #20]

ldr r3, [r7, #28]

subs r3, r2, r3

vmov s15, r3 @ int

vcvt.f64.s32 d16, s15

vldr.64 d18, .L10

vmul.f64 d16, d16, d18

vadd.f64 d16, d17, d16

vmov r2, r3, d16

movw r0, #:lower16:.LC7

movt r0, #:upper16:.LC7

bl printf

movs r3, #0

.L9:

mov r0, r3

adds r7, r7, #64

mov sp, r7

pop {r4, r5, r7, pc}

.L10:

.word -400107883

.word 1041313291

CalcSin:

push {r4, r5, r7, r8, r9, r10, fp, lr}

vpush.64 {d8}

sub sp, sp, #56

add r7, sp, #0

vstr.64 d0, [r7, #24]

strd r0, [r7, #16]

mov r2, #0

mov r3, #0

strd r2, [r7, #48]

vldr.64 d16, [r7, #24]

vldr.64 d17, .L16

vmul.f64 d17, d16, d17

vldr.64 d18, .L16+8

vdiv.f64 d16, d17, d18

vstr.64 d16, [r7, #24]

ldrd r2, [r7, #24]

strd r2, [r7, #40]

mov r2, #1

mov r3, #0

strd r2, [r7, #32]

b .L13

.L14:

vldr.64 d17, [r7, #48]

vldr.64 d16, [r7, #40]

vadd.f64 d16, d17, d16

vstr.64 d16, [r7, #48]

vldr.64 d17, [r7, #40]

vldr.64 d16, [r7, #24]

vmul.f64 d17, d17, d16

vldr.64 d16, [r7, #24]

vmul.f64 d16, d17, d16

vneg.f64 d8, d16

ldrd r2, [r7, #32]

adds r4, r2, #1

adc r5, r3, #0

ldrd r2, [r7, #32]

adds r8, r2, #2

adc r9, r3, #0

mul r2, r8, r5

mul r3, r4, r9

add r3, r3, r2

umull r10, fp, r4, r8

add r3, r3, fp

mov fp, r3

mov r0, r10

mov r1, fp

bl \_\_aeabi\_l2d

vmov d17, r0, r1

vdiv.f64 d16, d8, d17

vstr.64 d16, [r7, #40]

ldrd r2, [r7, #32]

adds r1, r2, #2

str r1, [r7, #8]

adc r3, r3, #0

str r3, [r7, #12]

ldrd r2, [r7, #8]

strd r2, [r7, #32]

.L13:

ldrd r2, [r7, #16]

adds r1, r2, r2

str r1, [r7]

adcs r3, r3, r3

str r3, [r7, #4]

ldrd r0, [r7]

ldrd r2, [r7, #32]

cmp r2, r0

sbcs r3, r3, r1

blt .L14

ldrd r2, [r7, #48]

vmov d16, r2, r3

vmov.f64 d0, d16

adds r7, r7, #56

mov sp, r7

vldm sp!, {d8}

pop {r4, r5, r7, r8, r9, r10, fp, pc}

.L16:

.word 1413753926

.word 1074340347

.word 0

.word 1080459264

**ПРИЛОЖЕНИЕ 3.** Ассемблерный листинг с оптимизацией -O1

CalcSin:

push {r3, r4, r5, r6, r7, r8, r9, lr}

vpush.64 {d8, d9, d10}

vldr.64 d16, .L7

vmul.f64 d0, d0, d16

vldr.64 d16, .L7+8

vdiv.f64 d9, d0, d16

adds r8, r0, r0

adc r9, r1, r1

cmp r8, #2

sbcs r3, r9, #0

blt .L4

vmov.f64 d16, d9

movs r4, #1

movs r7, #0

vmov.i64 d8, #0 @ float

.L3:

vadd.f64 d8, d8, d16

vmul.f64 d16, d9, d16

vnmul.f64 d10, d9, d16

adds r0, r4, #1

adc r2, r7, #0

adds r6, r4, #2

adc r5, r7, #0

mov r4, r6

mov r7, r5

mul r3, r0, r5

mla r3, r6, r2, r3

umull r0, r1, r0, r6

add r1, r1, r3

bl \_\_aeabi\_l2d

vmov d17, r0, r1

vdiv.f64 d16, d10, d17

cmp r6, r8

sbcs r5, r5, r9

blt .L3

.L1:

vmov.f64 d0, d8

vldm sp!, {d8-d10}

pop {r3, r4, r5, r6, r7, r8, r9, pc}

.L4:

vmov.i64 d8, #0 @ float

b .L1

.L7:

.word 1413753926

.word 1074340347

.word 0

.word 1080459264

.LC0:

.ascii "Bad input! Few arguments. Enter x and n in command "

.ascii "line.\000"

.LC1:

.ascii "Bad input! A lot of arguments. Enter x and n in com"

.ascii "mand line.\000"

.LC2:

.ascii "Timer resolution: %ld sec, %ld nanosec.\012\000"

.LC3:

.ascii "Call error clock\_getres!\000"

.LC4:

.ascii "Error: Invalid input for x: %s\012\000"

.LC5:

.ascii "Error: Invalid input for n: %s\012\000"

.LC6:

.ascii "%lf\012\000"

.LC7:

.ascii "Time takken: %lf sec.\012\000"

main:

push {r4, r5, r6, lr}

vpush.64 {d8}

sub sp, sp, #32

cmp r0, #2

ble .L18

mov r4, r1

cmp r0, #3

bgt .L19

add r1, sp, #24

movs r0, #4

bl clock\_getres

cmp r0, #0

bne .L13

ldr r2, [sp, #28]

ldr r1, [sp, #24]

movw r0, #:lower16:.LC2

movt r0, #:upper16:.LC2

bl printf

.L14:

movs r2, #10

add r1, sp, #4

ldr r0, [r4, #4]

bl strtol

vmov s16, r0 @ int

movs r2, #10

mov r1, sp

ldr r0, [r4, #8]

bl strtol

mov r5, r0

asrs r6, r0, #31

ldr r3, [sp, #4]

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

bne .L20

ldr r3, [sp]

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

bne .L21

add r1, sp, #16

movs r0, #4

bl clock\_gettime

mov r0, r5

mov r1, r6

vcvt.f64.s32 d0, s16

bl CalcSin

vmov r4, r5, d0

add r1, sp, #8

movs r0, #4

bl clock\_gettime

mov r2, r4

mov r3, r5

movw r0, #:lower16:.LC6

movt r0, #:upper16:.LC6

bl printf

ldr r3, [sp, #12]

ldr r2, [sp, #20]

subs r3, r3, r2

vmov s15, r3 @ int

vcvt.f64.s32 d17, s15

ldr r3, [sp, #8]

ldr r2, [sp, #16]

subs r3, r3, r2

vmov s15, r3 @ int

vcvt.f64.s32 d16, s15

vldr.64 d18, .L22

vmla.f64 d16, d17, d18

vmov r2, r3, d16

movw r0, #:lower16:.LC7

movt r0, #:upper16:.LC7

bl printf

movs r0, #0

.L9:

add sp, sp, #32

vldm sp!, {d8}

pop {r4, r5, r6, pc}

.L18:

movw r0, #:lower16:.LC0

movt r0, #:upper16:.LC0

bl printf

mov r0, #-1

b .L9

.L19:

movw r0, #:lower16:.LC1

movt r0, #:upper16:.LC1

bl printf

mov r0, #-1

b .L9

.L13:

movw r0, #:lower16:.LC3

movt r0, #:upper16:.LC3

bl perror

b .L14

.L20:

ldr r1, [r4, #4]

movw r0, #:lower16:.LC4

movt r0, #:upper16:.LC4

bl printf

mov r0, #-1

b .L9

.L21:

ldr r1, [r4, #8]

movw r0, #:lower16:.LC5

movt r0, #:upper16:.LC5

bl printf

mov r0, #-1

b .L9

.L22:

.word -400107883

.word 1041313291

**ПРИЛОЖЕНИЕ 4.** Ассемблерный листинг с самой быстрой у меня оптимизацией -Ofast

.LC0:

.ascii "Bad input! Few arguments. Enter x and n in command "

.ascii "line.\000"

.LC1:

.ascii "Bad input! A lot of arguments. Enter x and n in com"

.ascii "mand line.\000"

.LC2:

.ascii "Timer resolution: %ld sec, %ld nanosec.\012\000"

.LC3:

.ascii "Call error clock\_getres!\000"

.LC4:

.ascii "Error: Invalid input for x: %s\012\000"

.LC5:

.ascii "Error: Invalid input for n: %s\012\000"

.LC6:

.ascii "%lf\012\000"

.LC7:

.ascii "Time takken: %lf sec.\012\000"

main:

push {r4, r5, r6, r7, lr}

cmp r0, #2

vpush.64 {d8, d9, d10}

sub sp, sp, #36

ble .L16

cmp r0, #3

bne .L17

mov r4, r1

movs r0, #4

add r1, sp, #8

bl clock\_getres

cmp r0, #0

bne .L6

movw r0, #:lower16:.LC2

movt r0, #:upper16:.LC2

ldrd r1, r2, [sp, #8]

bl printf

.L7:

movs r2, #10

mov r1, sp

ldr r0, [r4, #4]

bl strtol

movs r2, #10

mov r3, r0

add r1, sp, #4

ldr r0, [r4, #8]

vmov s16, r3 @ int

bl strtol

ldr r3, [sp]

mov r6, r0

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

bne .L18

ldr r3, [sp, #4]

ldrb r5, [r3] @ zero\_extendqisi2

cmp r5, #0

bne .L19

add r1, sp, #16

movs r0, #4

bl clock\_gettime

vcvt.f64.s32 d16, s16

vldr.64 d17, .L20

asrs r7, r6, #31

adds r6, r6, r6

adcs r7, r7, r7

cmp r6, #2

vmul.f64 d16, d16, d17

sbcs r3, r7, #0

blt .L12

vmul.f64 d10, d16, d16

vmov.i64 d8, #0 @ float

movs r4, #1

.L11:

adds r0, r4, #1

vnmul.f64 d9, d16, d10

adc r2, r5, #0

adds r4, r4, #2

adc r5, r5, #0

vadd.f64 d8, d8, d16

mul r3, r0, r5

mla r3, r4, r2, r3

umull r0, r1, r0, r4

add r1, r1, r3

bl \_\_aeabi\_l2d

vmov d17, r0, r1

cmp r4, r6

vdiv.f64 d16, d9, d17

sbcs r3, r5, r7

blt .L11

.L10:

add r1, sp, #24

movs r0, #4

bl clock\_gettime

vmov r2, r3, d8

movw r0, #:lower16:.LC6

movt r0, #:upper16:.LC6

bl printf

ldr r2, [sp, #28]

ldrd r0, r3, [sp, #20]

ldr r1, [sp, #16]

subs r2, r2, r0

vmov s15, r2 @ int

subs r3, r3, r1

vldr.64 d18, .L20+8

vcvt.f64.s32 d17, s15

vmov s15, r3 @ int

movw r0, #:lower16:.LC7

movt r0, #:upper16:.LC7

vcvt.f64.s32 d16, s15

vmla.f64 d16, d17, d18

vmov r2, r3, d16

bl printf

movs r0, #0

.L1:

add sp, sp, #36

vldm sp!, {d8-d10}

pop {r4, r5, r6, r7, pc}

.L12:

vmov.i64 d8, #0 @ float

b .L10

.L17:

movw r0, #:lower16:.LC1

movt r0, #:upper16:.LC1

bl printf

.L3:

mov r0, #-1

b .L1

.L19:

ldr r1, [r4, #8]

movw r0, #:lower16:.LC5

movt r0, #:upper16:.LC5

bl printf

b .L3

.L18:

ldr r1, [r4, #4]

movw r0, #:lower16:.LC4

movt r0, #:upper16:.LC4

bl printf

b .L3

.L16:

movw r0, #:lower16:.LC0

movt r0, #:upper16:.LC0

bl printf

b .L3

.L6:

movw r0, #:lower16:.LC3

movt r0, #:upper16:.LC3

bl perror

b .L7

.L20:

.word -1571644252

.word 1066524486

.word -400107883

.word 1041313291

CalcSin:

vldr.64 d16, .L28

push {r3, r4, r5, r6, r7, lr}

adds r6, r0, r0

adc r7, r1, r1

cmp r6, #2

vmul.f64 d16, d0, d16

sbcs r3, r7, #0

vpush.64 {d8, d9, d10}

blt .L25

vmul.f64 d10, d16, d16

vmov.i64 d8, #0 @ float

movs r4, #1

movs r5, #0

.L24:

adds r0, r4, #1

vnmul.f64 d9, d16, d10

adc r2, r5, #0

adds r4, r4, #2

adc r5, r5, #0

vadd.f64 d8, d8, d16

mul r3, r0, r5

mla r3, r4, r2, r3

umull r0, r1, r0, r4

add r1, r1, r3

bl \_\_aeabi\_l2d

vmov d17, r0, r1

cmp r4, r6

vdiv.f64 d16, d9, d17

sbcs r3, r5, r7

blt .L24

vmov.f64 d0, d8

vldm sp!, {d8-d10}

pop {r3, r4, r5, r6, r7, pc}

.L25:

vmov.i64 d8, #0 @ float

vmov.f64 d0, d8

vldm sp!, {d8-d10}

pop {r3, r4, r5, r6, r7, pc}

.L28:

.word -1571644252

.word 1066524486

Ссылка на репозиторий с кодом: <https://github.com/dadashasha/nsu_evm/tree/main/lab4>