###### 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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Введение в архитектуру ARM»

студента 2 курса, группы 20204

**Дронова Дениса Юрьевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Кандидат технических наук

А. Ю. Власенко

Новосибирск 2021

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc91102937)

[ЗАДАНИЕ 3](#_Toc91102938)

[ОПИСАНИЕ РАБОТЫ 4](#_Toc91102939)

[ЗАКЛЮЧЕНИЕ 5](#_Toc91102940)

[Приложение 1. *Ассемблерный листинг arm\_O0* 6](#_Toc91102941)

[Приложение 2. *Ассемблерный листинг arm\_O3* 9](#_Toc91102942)

[Приложение 3. *Листинг программы sin(x)* 12](#_Toc91102943)

# ЦЕЛЬ

1. Знакомство с программной архитектурой ARM.

2. Анализ ассемблерного листинга программы для архитектуры ARM при разных уровнях компиляции.

# ЗАДАНИЕ

1. Изучить основы программной архитектуры ARM.

2. Для программы на языке Си (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры ARM, используя уровни оптимизации O0 и O3.

3. Проанализировать полученные листинги и сделать следующее:

• Сопоставить команды языка Си с машинными командами.

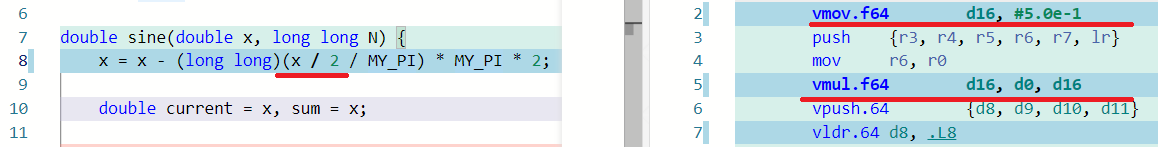
• Определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти).

• Описать и объяснить оптимизационные преобразования, выполненные компилятором.

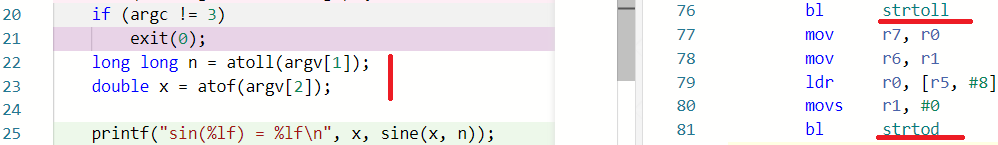
• Продемонстрировать использование ключевых особенностей архитектуры ARM на конкретных участках ассемблерного кода

# ОПИСАНИЕ РАБОТЫ

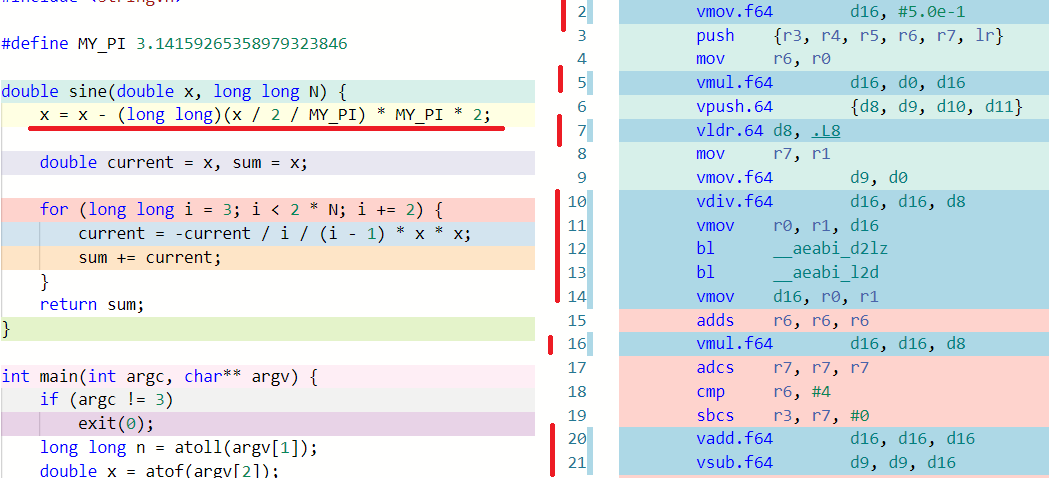
После компиляции программы, написанной на языке Си ключей оптимизации -O0, -O3, были получены ассемблерные листинги, приведенные в приложении, которые в последующем были проанализированы. Во время разбора были сопоставлены инструкции на языке Си с машинными командами, замечены ключевые особенности архитектуры ARM. После анализа каждой ассемблерной программы были составлены комментарии и обнаружены следующие оптимизации компилятора:

1. **Меньшее количество обращений в память (ldr, str)**
2. **Использование большего количества SIMD-регистров и операций с ними**
3. **Встраивание тела функции в код**
4. **Замена операторов на их более низкоуровневые аналоги**

*Умножение на ½ вместо деления на 2*

1. **Замена некоторых функций на их более универсальный аналог**

*Использование strtoll вместо atoll, strtod вместо atof, т. к. функции вида ato при неудачной конвертации строки в число возвращают 0, что может привести к undefined behavior*

1. **Изменение порядка выполнения инструкций**

# ЗАКЛЮЧЕНИЕ

Во время выполнения практической работы была изучена архитектура ARM: набор регистров, основные арифметико-логические команды, работа со стеком, вызов подпрограмм. На основе анализа ассемблерных листингов можно прийти к выводу о том, что при компиляции на уровне O0 программный код практически дословно переводится в набор команд на ассемблерном коде. При компиляции на уровне O3 программный код подвергается оптимизациям, приведенным описании работы.

# Приложение 1. *Ассемблерный листинг arm\_O0*

sine:

        ;параметры пришлм на регистрах r0, d0

        ;d(n) - SIMD registers

        push    {r4, r5, r7, r8, r9, r10, fp, lr} ;сохраняем значения регистров на стеке

        vpush.64        {d8, d9}

        sub     sp, sp, #40     ;sp = sp - 40 (выделение места)

        add     r7, sp, #0      ;r7 указывает на вершину стека

        vstr.64 d0, [r7, #8]    ;запись по адресу r7 + 8 значения лежащего в d0

                                ;[r7 + 8] = x

        strd    r0, [r7]        ;запись по адресу r7 значения лежащего в r0

                                ;[r7] = N

        vldr.64 d17, [r7, #8]   ;d17 = x

        vmov.f64        d18, #2.0e+0 ;d18 = 2 (константа)

        vdiv.f64        d16, d17, d18   ;d16 = x / 2

        vldr.64 d17, .L5        ;d17 = MY\_PI

        vdiv.f64        d18, d16, d17   ;d18 = (x/2) / MY\_PI

        vmov    r0, r1, d18     ;запись в r0, r1 двух слов из d18

        bl      \_\_aeabi\_d2lz ;конвертация double в long long

        mov     r2, r0  ;r2 = r0 //

        mov     r3, r1  ;r3 = r1 //     Для чего эти махинации???

        mov     r0, r2  ;r0 = r2 //

        mov     r1, r3  ;r1 = r3 //

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

        vmov    d17, r0, r1 ;d17 = r0 r1 (записали два регистра в SIMD регистр)

        vldr.64 d16, .L5        ;d16 = MY\_PI

        vmul.f64        d16, d17, d16   ;d16 = d17(previous result) \* MY\_PI

        vadd.f64        d16, d16, d16   ;d16 = d16 \* 2

        vldr.64 d17, [r7, #8] ;d17 = x

        vsub.f64        d16, d17, d16 ;d16 = x - d16 (previous result)

        vstr.64 d16, [r7, #8]   ;[r7 + 8] = result (x)

        ldrd    r2, [r7, #8]    ;r2 = x

        strd    r2, [r7, #32]   ;запись x по адресу r7 + 32 (current = x)

        ldrd    r2, [r7, #8]    ;r2 = x

        strd    r2, [r7, #24]   ;запись x по адресу r7 + 24 (sum = x)

        mov     r2, #3  ;r2 = 3 (constant) (i = 3)

        mov     r3, #0  ;r3 = 0 (constant)

        strd    r2, [r7, #16]   ;[r7 + 16] = 3 (сохранили значение i)

        b       .L2

.L3:

        vldr.64 d16, [r7, #32] ;d16 = current

        vneg.f64        d9, d16 ;d9 = -d16 (-current)

        ldrd    r0, [r7, #16]   ;r0 = значение i (3 - на первой итерации)

        bl      \_\_aeabi\_l2d     ;конвертация long -> double

        vmov    d16, r0, r1     ;d16 = результат конвертации (i)

        vdiv.f64        d8, d9, d16 ;d8 = -current / i

        ldrd    r2, [r7, #16]   ;r2 = i(3)

        adds    r8, r2, #-1     ;r8 = i(3) - 1;

        adc     r9, r3, #-1     ;r9 = r3(flag) - 1

        mov     r0, r8          ;r0 = i - 1

        mov     r1, r9

        bl      \_\_aeabi\_l2d     ;конвертация long -> double для выполнения последующего деления

        vmov    d16, r0, r1     ;d16 = r0 r1

        vdiv.f64        d17, d8, d16    ;d17 = -current / i / (i - 1)

        vldr.64 d16, [r7, #8]   ;d16 = x

        vmul.f64        d16, d17, d16   ;d16 = (-current / i / (i - 1)) \* x

        vldr.64 d17, [r7, #8]   ;d17 = x

        vmul.f64        d16, d17, d16 ;d16 = d16 \* x

        vstr.64 d16, [r7, #32]  ;current ([r7 + 32]) = d16

        vldr.64 d17, [r7, #24]  ;d17 = sum([r7 + 24])

        vldr.64 d16, [r7, #32]  ;d16 = current

        vadd.f64        d16, d17, d16   ;sum += current

        vstr.64 d16, [r7, #24]  ;[r7 + 24] = sum

        ldrd    r2, [r7, #16]   ;r2 = i

        adds    r4, r2, #2      ;r4 = i(r2) + 2

        adc     r5, r3, #0

        strd    r4, [r7, #16]   ;[r7 + 16] = r4 (new i) перезаписали значение i

.L2:

        ldrd    r2, [r7]        ;r2 = N

        adds    r10, r2, r2     ;r10 = 2 \* N

        ;fp - флаг

        ;long long лежит в двух 32-битных регистрах r2 r3 (r2 - младшые биты, r3 - старшие)

        ;adc - происходит сложение с учетом того, если при сложении r2 произошел перенос

        adc     fp, r3, r3   ;fp = r3 + r3 + carry flag

        mov     r0, r10      ;r0 = 2N

        mov     r1, fp       ;r1 = (flag)

        ldrd    r2, [r7, #16]   ;r2 = 3

        cmp     r2, r0          ;сравнение i==3 и 2N

        sbcs    r3, r3, r1   ;r3 = r3 - r1 (если carry flag - set)

        blt     .L3     ;branch lower than...

        ldrd    r2, [r7, #24]   ;r2 = sum ([r7 + 24])

        vmov    d16, r2, r3

        vmov.f64        d0, d16

        adds    r7, r7, #40

        mov     sp, r7

        vldm    sp!, {d8-d9}

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

.L5:

        .word   1413754136

        .word   1074340347

.LC0:

        .ascii  "sin(%lf) = %lf\012\000"

main:

        ;при вызове bl адрес следующей команды сохраняется в lr

        ;lr хранит в себе адрес возврата из подпрограммы (адрес следующей команды)

        ;Входные параметры сохраняются в r0, r1 (argc, argv)

        push    {r7, lr}        ;Помещаем в стек (обращение в память) r7 и LinkRegister (r13),

                                ;чтобы затем вернуть их в исходное состояние

        sub     sp, sp, #32     ;sp = sp - 32 (выделение места на стеке)

        add     r7, sp, #8      ;r7 = sp + 8

        str     r0, [r7, #4]    ;записать по адресу r7 + 4 содержимое r0 (argc)

        str     r1, [r7]        ;записать по адресу r7 содержимое r1 (argv)

        ldr     r3, [r7, #4]    ;загрузить в r3 содержимое адреса r7 + 4 (r3 = argc)

        cmp     r3, #3          ;argc == 3 ?

        beq     .L8             ;если argc = 3, то перейти к L8

        movs    r0, #0          ;если argc != 3, то r0 = 0 (код возврата)

        bl      exit

.L8:

        ldr     r3, [r7]        ;r3 = адрес argv

        adds    r3, r3, #4      ;r3 += 4

        ldr     r3, [r3]        ;поместить в r3 содержимое, лежащее по адресу, храняещемуся в r3

                                ;r3 = argv[1]

        mov     r0, r3          ;r0 = argv[1] (для вызова подограммы atoll)

        bl      atoll

        strd    r0, [r7, #16]   ;записать по адресу r7 + 16 результат выполнения atoll

                                ;[r7 + 16] = n

        ldr     r3, [r7]

        adds    r3, r3, #8

        ldr     r3, [r3]

        mov     r0, r3          ;r0 = argv[2]

        bl      atof

        vstr.64 d0, [r7, #8]    ;записать из d0(64 bit) в адрес r7 + 8

                                ;[r7 + 8] = x

        ldrd    r0, [r7, #16]   ;r0 = n

        vldr.64 d0, [r7, #8]    ;d0 = x

        bl      sine

        vmov.f64        d16, d0 ;загрузить из d0 в d16

        vstr.64 d16, [sp]       ;записать из d16 на адрес sp

        ldrd    r2, [r7, #8]    ;r2 = x

        movw    r0, #:lower16:.LC0

        movt    r0, #:upper16:.LC0

        bl      printf

        movs    r3, #0  ;r3 = 0 (код возврата)

        mov     r0, r3  ;r0 = r3(0)

        adds    r7, r7, #24

        mov     sp, r7

        pop     {r7, pc} ;достаем из стека начальное значение r7 (до вызова main)

                         ;и достаем адрес возврата и помещаем в pc (program counter)

                         ;pc - адрес следующей команды для исполнения

                ;пищем в pc какой-либо адрес -- аналог jump

# Приложение 2. *Ассемблерный листинг arm\_O3*

;встривание тела функции в код

sine:

        vmov.f64        d16, #5.0e-1 ;d16 = 0.5

        push    {r3, r4, r5, r6, r7, lr} ;сохранение значений регистров

        mov     r6, r0 ;r6 = x(r0)

        vmul.f64        d16, d0, d16 ;d16 = x(d0) \* 0.5(d16)

        vpush.64        {d8, d9, d10, d11} ;сохранение значений регистров

        vldr.64 d8, .L8  ;d8 = MY\_PI

        mov     r7, r1 ;r7 = N (r1)

        vmov.f64        d9, d0 ;d9 = x

        vdiv.f64        d16, d16, d8 ;d16 = (x \* 0.5) / MY\_PI

        vmov    r0, r1, d16 ;unpack SIMD register

        bl      \_\_aeabi\_d2lz

        bl      \_\_aeabi\_l2d

        vmov    d16, r0, r1

        adds    r6, r6, r6 ;r6 = 2 \* N

        vmul.f64        d16, d16, d8 ;d16 = ((x \* 0.5) / MY\_PI) \* MY\_PI

        adcs    r7, r7, r7

        cmp     r6, #4 ;сравнение 2 \* N с 4

        sbcs    r3, r7, #0

        vadd.f64        d16, d16, d16 ;d16 \*= 2;

        vsub.f64        d9, d9, d16 ;d9 (x) = x - (((x \* 0.5) / MY\_PI) \* 2) \* MY\_PI

        blt     .L4 ;if 2N < 4 -> goto L4

        vmov.f64        d10, d9 ;sum = x

        vmov.f64        d16, d9 ;current = x

        movs    r4, #3  ;r4 = 3

        movs    r5, #0 ;r5 = 0

.L3:

        vneg.f64        d8, d16 ;d8 = -current

        mov     r0, r4  ;r0 = 3

        mov     r1, r5  ;r1 = 0

        bl      \_\_aeabi\_l2d

        vmov    d16, r0, r1 ;d16 = 3, 0

        subs    r3, r4, #1 ;r3 = i(3) - 1

        vdiv.f64        d11, d8, d16 ;d11 = -current / i(d16)

        sbc     r1, r5, #0 ;r1 = r5 - 0

        mov     r0, r3 ;r0 = (i - 1)

        bl      \_\_aeabi\_l2d

        vmov    d16, r0, r1 ;d16 = (i - 1), 0

        adds    r4, r4, #2 ;i(r4) += 2

        adc     r5, r5, #0

        cmp     r4, r6 ;comparison i and 2N

        sbcs    r3, r5, r7

        vdiv.f64        d16, d11, d16 ;d16 = (-current / i(d16)) / (i - 1)

        vmul.f64        d16, d16, d9 ;d16 = (-current / i(d16)) / (i - 1) \* x

        vmul.f64        d16, d16, d9 ;current (d16) = (-current / i(d16)) / (i - 1) \* x \* x

        vadd.f64        d10, d10, d16 ;sum = sum + current

        blt     .L3 ;if i < 2N --> goto .L3

        vmov.f64        d0, d10 ;d0 = sum

        vldm    sp!, {d8-d11}

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

.L4:

        vmov.f64        d10, d9 ;sum = x

        vmov.f64        d0, d10 ;d10 = sum(x)

        vldm    sp!, {d8-d11}

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

.L8:

        .word   1413754136

        .word   1074340347

.LC0:

        .ascii  "sin(%lf) = %lf\012\000"

main:

        push    {r4, r5, r6, r7, lr} ;Помещаем в стек регистры r4 - r7 и LinkRegister (r13),

                                     ;чтобы затем вернуть их в исходное состояние

        cmp     r0, #3  ;r0(argc) == 3 ?

        vpush.64        {d8, d9, d10, d11, d12} ;помещаем в стек регистры d8 - d12

        sub     sp, sp, #12 ;наращивание стека

        beq     .L11    ;если argc = 3, то переходим к L11

        movs    r0, #0 ;если argc != 3, то выход с кодом 0

        bl      exit

.L11:

        mov     r5, r1 ;r5 = r1(argv)

        movs    r2, #10 ;r2 =10

        mov     r4, r0 ;r4 = argc (3)

        ldr     r0, [r1, #4] ;r0 = argv[1]

        movs    r1, #0  ;r1 = 0

        vldr.64 d10, .L17 ;d10 = MY\_PI - заранее подготовили константу

        bl      strtoll

        mov     r7, r0  ;r7(n) = result of strtoll

        mov     r6, r1  ;r6 = r1(0)

        ldr     r0, [r5, #8] ;r0 = [r5 + 8] (argv[2])

        movs    r1, #0  ;r1 = 0

        bl      strtod  ;x - r0(result of strtod)

        vmov.f64        d16, #5.0e-1 ;d16 = 0.5 (constant) - заранее подготовили константу

        vmov.f64        d9, d0 ;d9 = x

        vmul.f64        d16, d0, d16 ;d16 = x \* 0.5

        vdiv.f64        d16, d16, d10 ;d16 = (x \* 0.5) / MY\_PI

        vmov    r0, r1, d16 ;распаковали в два регистра

        bl      \_\_aeabi\_d2lz    ;convert double to long

        bl      \_\_aeabi\_l2d     ;convert long to double

        vmov    d16, r0, r1     ;d16 = r0 r1

        adds    r7, r7, r7 ;i = 2 \* N

        vmul.f64        d10, d16, d10 ;d10 = d16 ((x \* 0.5) / MY\_PI) \* MY\_PI

        adcs    r6, r6, r6

        cmp     r7, #4 ;сравнение 2N с 4

        sbcs    r3, r6, #0

        vadd.f64        d10, d10, d10 ;d10 \*= 2

        vsub.f64        d10, d9, d10    ;d10 = x - d10

        blt     .L14    ;если 2N < 4 то в основной цикл не зайдем --> выход из подпрограммы

        vmov.f64        d11, d10 ;sum(d11) = x(d10)

        vmov.f64        d16, d10 ;current (d16) = x(d10)

        movs    r5, #0  ;r5 = 0

.L13:

        vneg.f64        d8, d16 ;d8 = -current

        mov     r0, r4 ;r0 = 3

        mov     r1, r5

        bl      \_\_aeabi\_l2d

        vmov    d16, r0, r1 ;d16 = 3, 0

        subs    r3, r4, #1 ;r3 = i(3) - 1

        vdiv.f64        d12, d8, d16 ;d12 = -current / i(d16)

        sbc     r1, r5, #0

        mov     r0, r3 ;r0 = (i - 1)

        bl      \_\_aeabi\_l2d

        vmov    d16, r0, r1 ;d16 = (i - 1), 0

        adds    r4, r4, #2 ;i(r4) += 2

        adc     r5, r5, #0

        cmp     r4, r7 ;comparison i and 2N

        sbcs    r3, r5, r6

        vdiv.f64        d16, d12, d16 ;d16 = (-current / i(d16)) / (i - 1)

        vmul.f64        d16, d16, d10 ;d16 = (-current / i(d16)) / (i - 1) \* x

        vmul.f64        d16, d10, d16 ;current (d16) = (-current / i(d16)) / (i - 1) \* x \* x

        vadd.f64        d11, d11, d16 ;sum = sum + current

        blt     .L13 ;if i < 2N --> goto .L3

.L12:

        vmov    r2, r3, d9 ;{r2, r3} = x - параметры функции printf

        vstr.64 d11, [sp]

        movw    r0, #:lower16:.LC0

        movt    r0, #:upper16:.LC0

        bl      printf

        ;завершение программы

        movs    r0, #0

        add     sp, sp, #12

        vldm    sp!, {d8-d12}

        pop     {r4, r5, r6, r7, pc}

.L14:

        vmov.f64        d11, d10 ;sum(d11) = x(d10) т.к в основной цикл функции не заходим,

                                 ;но вернуть sum необходимо

        b       .L12

.L17:

        .word   1413754136

        .word   1074340347

# Приложение 3. *Листинг программы sin(x)*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MY\_PI 3.14159265358979323846

double sine(double x, long long N) {

    x = x - (long long)(x / 2 / MY\_PI) \* MY\_PI \* 2;

    double current = x, sum = x;

    for (long long i = 3; i < 2 \* N; i += 2) {

        current = -current / i / (i - 1) \* x \* x;

        sum += current;

    }

    return sum;

}

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

    if (argc != 3)

        exit(0);

    long long n = atoll(argv[1]);

    double x = atof(argv[2]);

    printf("sin(%lf) = %lf\n", x, sine(x, n));

    return 0;

}