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

студентки 2 курса, 24202 группы

Корсун Дарьи Андреевны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
В.А.Перепёлкин

Новосибирск 2025

СОДЕРЖАНИЕ

ЦЕЛЬ

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

ЗАДАНИЕ

Изучить основы программной архитектуры ARM.
2. Для программы на языке Си (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры ARM, используя различные уровни комплексной оптимизации.
3. Проанализировать полученные листинги.

ОПИСАНИЕ РАБОТЫ

Для анализ ассемблерного листинга программы для архитектуры x86/x86-64 выбрана программа, реализующая подсчет значения синуса через разложение в ряд Тейлора. Ниже представлена программа, реализующая вычисления функции sin x с помощью разложения в степенной ряд по первым N членам этого ряда:

$$\sin x = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{(-1)^{n-1}}{(2n-1)!} x^{2n-1} + \cdots$$

Область сходимости ряда: $-\infty \le x \le \infty$

Код программы

```c
#include <math.h>
#include <stdio.h>

double CalculateSin(double x, long N) {
  double term = x;
  double result = x;
  for (long n = 2; n <= N; ++n) {
    term *= -(x * x) / ((2 * n - 1) * (2 * n - 2));
    result += term;
  }
  return result;
}
```

Проанализируем полученные на ассемблере код:

```
        push    {r7}                    // Сохраняем регистр
        sub     sp, sp, #44             // Выделяем память на стеке (44 байта)
        add     r7, sp, #0              // r7 = указатель на начало фрейма
        vstr.64 d0, [r7, #8]            // x (d0) в стек
        str     r0, [r7, #4]            // N (r0) в стек
        ldrd    r2, [r7, #8]            // x в r2:r3
        strd    r2, [r7, #32]           // term = x
        ldrd    r2, [r7, #8]            // Снова загружаем x
        strd    r2, [r7, #24]           // result = x
        movs    r3, #2                  // n = 2
        str     r3, [r7, #20]           // n в стек
        b       .L2                     // проверка условия
.L3:
        vldr.64 d16, [r7, #8]          // d16 = x
        vmul.f64        d16, d16, d16  // d16 = x²
        vneg.f64        d18, d16        // d18 = -x²
        ldr     r3, [r7, #20]           // r3 = n
        subs    r3, r3, #1              // r3 = n - 1
        ldr     r2, [r7, #20]           // r2 = n
        lsls    r2, r2, #1              // r2 = 2k
        subs    r2, r2, #1              // r2 = 2k - 1
        mul     r3, r2, r3              // r3 = (2k-1) * (n-1)
        lsls    r3, r3, #1              // r3 = 2 * (2k-1) * (n-1) = (2k-1)*(2k-2)
        vmov    s15, r3 @ int           // Переносим в FPU
        vcvt.f64.s32    d17, s15        // Конвертируем в double -> d17 = (2k-1)!
        vdiv.f64        d16, d18, d17   // d16 = -x² / ((2k-1)*(2k-2))
        vldr.64 d17, [r7, #32]          // d17 =(term{n-1})
        vmul.f64        d16, d17, d16   // d16 = T{n-1}*(-x²/((2k-1)*(2k-2)))= T_k
        vstr.64 d16, [r7, #32]          // d16 —>  term
        vldr.64 d17, [r7, #24]          // d17 = result
        vldr.64 d16, [r7, #32]          // d16 <— term
        vadd.f64        d16, d17, d16   // result += term
```

```
        vstr.64 d16, [r7, #24]          // Cd16 -> result
        ldr     r3, [r7, #20]           // r3 = n
        adds    r3, r3, #1              // n++
        str     r3, [r7, #20]           // Сохраняем n
.L2:
        ldr     r2, [r7, #20]           // Загружаем n
        ldr     r3, [r7, #4]            // Загружаем N
        cmp     r2, r3                  // Сравниваем n и N
        ble     .L3                      // Если n <= N, продолжаем цикл
        ldrd    r2, [r7, #24]           // Загружаем результат в r2:r3
        vmov    d16, r2, r3             // Переносим в d16
        vmov.f64        d0, d16         // Возвращаем результат через d0
        adds    r7, r7, #44             // Восстанавливаем указатель стека
        mov     sp, r7
        pop     {r7}                    // Восстанавливаем регистр
        bx      lr                      // Возврат  из функции



CalculateSin(double, long):
        vmov.f64        d17, d0          // d17 = x
        cmp     r0, #1                   // if (n <= 1)
        ble     .L4                      // Да -> возвращаем x
        vnmul.f64       d19, d0, d0      // d19 = -(x * x) = -x²
        movs    r1, #3                   // r1 = 3 (первый знаменатель: 2*2-1=3)
        movs    r2, #1                   // r2 = 1 (счётчик итераций n)
.L3:
        mul     r3, r1, r2               // r3 = (2n+1) * n
        adds    r2, r2, #1               // n++
        adds    r1, r1, #2               // r1 += 2 (готовим следующий знаменатель)
        cmp     r0, r2                   // Сравниваем N и n
        lsl     r3, r3, #1               // r3 = 2 * (2n+1) * n = (2n+1)*2n
        vmov    s15, r3 @ int            // Переносим в FPU
        vcvt.f64.s32    d16, s15         // Конвертируем в double -> d16 = (2n+1)!
        vdiv.f64        d18, d19, d16    // d18 = -x² / ((2n+1)*2n)
        vmul.f64        d17, d17, d18    // d17 = Term *(-x²/((2n+1)*2n)) = result
        vadd.f64        d0, d0, d17      // result += term
        bne     .L3                      // Если n != N -> continue
        bx      lr                       // Возврат из функции
.L4:
        bx      lr                       // Возврат из функции
```

Особенности архитектуры ARM

1) Комбинированные операции —> позволяет выполнять несколько арифметических операций за одну команду

2) Сдвиг может выполняться в рамках другой операции без дополнительных тактовых затрат

3) Возможность работы с парой регистров одновременно, что дает удвоение производительности

Различия между уровнями оптимизации:

1) В уровне оптимизации -O3 было убрано избыточное обращение к памяти в пользу выполнения всех функций на регистре, что ускоряет работу программы, так как обращение к памяти очень времяёмко

2) В уровне оптимизации -O3 было использовано предвычисление —> $-x^2$ был посчитан до тела цикла, что позволила на каждой итерации экономить время на подсчете этого параметра

3) В уровне оптимизации -O3 засчет оптимизации происходит сокращение кол-во затрачиваемых тактов за счет комбинированных операций —> vnmul.f64 (умножение и накладывания минуса за одну операцию)

4) В уровне оптимизации -O3 было замечен паттерн на возможность выхода из функции без выполнения проверок на счетчик if (n <= 1)

ЗАКЛЮЧЕНИЕ

Было проведено знакомство с программной архитектурой ARM. А также анализ ассемблерного листинга программы для архитектуры ARM.

Были сделаны выводы о преимуществе ARM архитектуры за счет ускорения и оптимизации многих функций за счет более сложных и в тоже время продуманных инструкций, что позволяет уменьшить итоговое время, требуемое процессором на выполнения программы.