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 Бородина Артёма Максимовича 2 курса, 19205 группы

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н, доцент
А.Ю. Власенко

Новосибирск 2020

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Знакомство с программной архитектурой ARM.

2. Анализ ассемблерного листинга программы для архитектуры ARM.

ЗАДАНИЕ

1 вариант:

Алгоритм вычисления числа π с помощью разложения в ряд (ряд Грегори-Лейбница) по формуле Лейбница N первых членов ряда.

1. 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

4. Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

1. Изучены основы архитектуры ARM, группы регистров, доступные программисту, сопроцессор и векторные расширения.

2. Были сделаны 3 листинга: код программы, вычисляющей число π, код программы на ассемблере, скомпилированной с ключом -O0 и код программы на ассемблере, скомпилированной с ключом -O2.

3. Для просмотра кода программы на ассемблере был использован ресурс godbolt.org. Компиляция проводилась для ARM64. (Листинги можно посмотреть по ссылке: https://godbolt.org/z/9nz6Y6)

4. При сравнении двух листингов на ассемблере становится заметно, что программа, скомпилированная с ключом -O2, требует гораздо меньше памяти для переменных и делает расчеты более эффективно (например, использует регистры для более быстрого доступа к данным или избегает хранения ненужных значений в случае с clock), по сравнению с программой, скомпилированной с ключом -O0. Так же компилятор с ключом -O2 добавил проверку на 0 в функции **LeibnizFormula(int)**, что позволило избежать заход в цикл для расчета числа π, т.к. после инициализации переменных цикл пришлось бы сразу завершать.

ЗАКЛЮЧЕНИЕ

После изучения архитектуры ARM, были рассмотрены программы на ассемблере, скомпилированные с ключами -O0 и -O2, и проведено последующее сравнение для поиска различий разных уровней оптимизации.

Приложение 1. *Код программы*

```
#include <cmath>
#include <cstdlib>
#include <ctime>

long double LeibnizFormula(int n) {
   long double pi = 0;
   for (int i = 0; i < n; ++i) {
      long double appendix  = 1 / (long double) (2 * i + 1);
      pi += appendix * pow(-1, i);
   }
   pi *= 4;
   return pi;
}


int main() {
   int N = 1000000;
   long double startTime = clock();
   long double pi = LeibnizFormula(N);
   long double endTime = clock();
   long double runTIme = endTime - startTime;
   return 0;
}
```

Приложение 2. *Ассемблерный код (ключ -O0)*

```
LeibnizFormula(int):
        stp     x29, x30, [sp, -80]!    // Сохранить указатель на предыдущий стек и sp -= 80
        mov     x29, sp                 // База нового стека = вершина старого
        str     w0, [sp, 28]            // Сохранить n в стек
        stp     xzr, xzr, [sp, 64]      // Сохранить pi = 0 в стек
        str     wzr, [sp, 60]           // Сохранить i = 0 в стек
.L3:
        ldr     w1, [sp, 60]            // Загрузить в w1 i
        ldr     w0, [sp, 28]            // Загрузить в w0 n
        cmp     w1, w0                  // Сравнить i и n
        bge     .L2                     // Переход, если i >= n
        ldr     w0, [sp, 60]            // Загрузить в w0 i
        lsl     w0, w0, 1               // Сдвинуть w0 на 1 бит влево (i *= 2)
        add     w0, w0, 1               // Добавить к w0 1 (i = 2 * i + 1)
        bl      __floatsitf             // Вызов подпрограммы __floatditf
        mov     v1.16b, v0.16b          // q1 = q0
        adrp    x0, .LC0                // x0 = 0x3FFF0000
        add     x0, x0, :lo12:.LC0      // x0 = 0x3FF00000 (= 1) (:lo12: берет первые 12 бит)
        ldr     q0, [x0]                // Загрузить в q0 x0
        bl      __divtf3                // Вызов подпрограммы __divtf3 (q0 = 1 / 2 * i + 1)
        str     q0, [sp, 32]            // Сохранить q0 в стек (appendix = q0)
        ldr     w0, [sp, 60]            // Загрузить в w0 i
        scvtf   d0, w0                  // Сменить тип на double-precision ((double)i)
        fmov    d1, d0                  // Сохранить в d1 (double)i
        fmov    d0, -1.0e+0             // d0 = -1.0
        bl      pow                     // Вызов подпрограммы pow (q0 = pow(-1, i))
        bl      __extenddftf2           // Вызов подпрограммы __extenddftf2
        ldr     q1, [sp, 32]            // Загрузить в q1 appendix
        bl      __multf3                // Вызов подпрограммы __multf3 (q0 *= q1)
        mov     v1.16b, v0.16b          // q1 = q0
        ldr     q0, [sp, 64]            // Загрузить в q0 pi
        bl      __addtf3                // Вызов подпрограммы __addtf3 (q0 += q1)
        str     q0, [sp, 64]            // Сохранить pi в стек
        ldr     w0, [sp, 60]            // Загрузить в w0 i
        add     w0, w0, 1               // Добавить к w0 1 (++i)
        str     w0, [sp, 60]            // Сохранить i в стек
        b       .L3
.L2:
        adrp    x0, .LC1                // x0 = 0x40100000
        add     x0, x0, :lo12:.LC1      // x0 = 0x40100000 (= 4)
        ldr     q1, [x0]                // Загрузить в q1 x0
        ldr     q0, [sp, 64]            // Загрузить в q0 pi
        bl      __multf3                // Вызов подпрограммы __multf3 (q0 *= q1)
        str     q0, [sp, 64]            // Сохранить в стек pi
        ldr     q0, [sp, 64]            // Загрузить в q0 pi
        ldp     x29, x30, [sp], 80      // Восстановить указатели на предыдущий стек
        ret                             // Возврат из подпрограммы
main:
        stp     x29, x30, [sp, -96]!    // Сохранить указатель на предыдущий стек и sp -= 96
        mov     x29, sp                 // База нового стека = вершина старого
        mov     w0, 16960               // w0 = 16960 (0x00004240)
        movk    w0, 0xf, lsl 16         // w0 = 0x00004240 + 0xF0000 = 0x000F4240 (N = 1000000)
        str     w0, [sp, 92]            // Сохранить N в стек
        bl      clock                   // Вызов подпрограммы clock
        bl      __floatditf             // Вызов подпрограммы __floatditf
        str     q0, [sp, 64]            // Сохранить startTime в стек
        ldr     w0, [sp, 92]            // Загрузить из памяти N
        bl      LeibnizFormula(int)     // Вызов подпрограммы LeibnizFormula
        str     q0, [sp, 48]            // Сохранить pi в стек
        bl      clock                   // Вызов подпрограммы clock
        bl      __floatditf             // Вызов подпрограммы __floatditf
        str     q0, [sp, 32]            // Сохранить endTime в стек
        ldr     q1, [sp, 64]            // Загрузить в q1 startTime
        ldr     q0, [sp, 32]            // Загрузить в q0 endTime
        bl      __subtf3                // Вызов подпрограммы __subtf3 (вычитание)
        str     q0, [sp, 16]            // Сохранить runTime = endTime - startTime в стек
        mov     w0, 0                   // return 0
        ldp     x29, x30, [sp], 96      // Восстановить указатели на предыдущий стек
        ret                             // Возврат из подпрограммы
.LC0:
        .word   1073676288              // 0x3FFF0000
.LC1:
        .word   1073807360              // 0x40100000
```

Приложение 3. *Ассемблерный код (ключ -O2)*

```
LeibnizFormula(int):
        cmp     w0, 0                   // Сравнить w0 с 0 (N == 0)
        ble     .L4                     // Переход, если N = 0
        movi    v2.2d, #0               // Записать в регистр q20 (.2d = 2Dword = Qword) (long double pi = 0)
        stp     x29, x30, [sp, -80]!    // Сохранить указатели на предыдущий стек и sp -= 80
        mov     x29, sp                 // База нового стека = вершина старого
        str     x21, [sp, 32]           // Сохранить x21 в стек
        mov     w21, w0                 // w21 = w0 (n)
        stp     x19, x20, [sp, 16]      // Сохранить x19, x20 в стек
        mov     w20, 1                  // w20 = 1
        mov     w19, 0                  // w19 = 0 (i)
        str     d8, [sp, 40]            // Сохранить d8 в стек
.L3:
        scvtf   d1, w19                 // Сменить тип на double-precision ((double)i)
        fmov    d0, -1.0e+0             // d0 = -1.0
        str     q2, [sp, 64]            // Сохранить q2 в стек pi
        add     w19, w19, 1             // w19 += 1 (++i)
        bl      pow                     // Вызов подпрограммы pow (q0 = pow(-1, i))
        fmov    d8, d0                  // d8 = d0
        mov     w0, w20                 // w0 = w20
        add     w20, w20, 2             // w20 += 2
        bl      __floatsitf             // Вызов подпрограммы __floatditf
        mov     v1.16b, v0.16b          // q1 = 0 (сохранить pow(-1, i))
        adrp    x0, .LC0                // x0 = 0x3FFF0000
        add     x0, x0, :lo12:.LC0      // x0 = 0x3FF00000 (= 1) (:lo12: берет первые 12 бит)
        ldr     q0, [x0]                // Загрузить в q0 x0
        bl      __divtf3                // Вызов подпрограммы __divtf3 (q0 = 1 / 2 * i + 1)
        str     q0, [sp, 48]            // Сохранить appendix в стек (appendix = q0)
        fmov    d0, d8                  // d0 = d8
        bl      __extenddftf2           // Вызов подпрограммы __extenddftf2
        mov     v1.16b, v0.16b          // q1 = q0
        ldr     q4, [sp, 48]            // Загрузить в q4 appendix
        mov     v0.16b, v4.16b          // q0 = q4
        bl      __multf3                // Вызов подпрограммы __multf3 (appendix * pow(-1, i))
        ldr     q2, [sp, 64]            // Загрузить в q2 pi
        mov     v1.16b, v0.16b          // q1 = q0
        mov     v0.16b, v2.16b          // q0 = q2
        bl      __addtf3                // Вызов подпрограммы __addtf3 (q0 += q1)
        cmp     w21, w19                // Сравнить w21 и w19 (i == n)
        mov     v2.16b, v0.16b          // q2 = q0 (сохранить pi)
        bne     .L3                     // Переход, если не равны
        adrp    x0, .LC1                // x0 = 0x40010000
        add     x0, x0, :lo12:.LC1      // x0 = 0x40100000 (= 4)
        ldr     q1, [x0]                // Загрузить в q1 x0
        bl      __multf3                // Вызов подпрограммы __multf3 (q0 *= q1) (pi *= 4)
        ldp     x19, x20, [sp, 16]      // Восстановить значения x19 и x20
        ldr     x21, [sp, 32]           // Восстановить значение x21
        ldr     d8, [sp, 40]            // Восстановить значение d8
        ldp     x29, x30, [sp], 80      // Восстановить указатели на предыдущий стек
        ret                             // Возврат из подпрограммы
.L4:
        movi    v0.2d, #0               // Записать в регистр v0 0
        ret                             // Возврат из подпрограммы
main:
        stp     x29, x30, [sp, -16]!    // Сохранить указатели на предыдущий стек и sp -= 16
        mov     x29, sp                 // База нового стека = вершина старого
        bl      clock                   // Вызов подпрограммы clock
        mov     w0, 16960               // w0 = 16960 (0x00004240)
        movk    w0, 0xf, lsl 16         // w0 = 0x00004240 + 0xF0000 = 0x000F4240 (N = 1000000)
        bl      LeibnizFormula(int)     // Вызов подпрограммы LeibnizFormula
        bl      clock                   // Вызов подпрограммы clock
        mov     w0, 0                   // return 0
        ldp     x29, x30, [sp], 16      // Восстановить указатели на предыдущий стек
        ret                             // Возврат из подпрограммы
.LC0:
        .word   1073676288              // 0x3FFF0000
.LC1:
        .word   1073807360              // 0x40010000
```