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, 21211 группы

Петрова Сергея Евгеньевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Антон Юрьевич Кудинов

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

- *Знакомство с программной архитектурой ARM;*
- *Анализ ассемблерного листинга программы для архитектуры ARM;*

ЗАДАНИЕ

1. *Изучить основы программной архитектуры ARM.*
2. *Написать программу на языке С, которая реализует алгоритм вычисления числа π методом Монте-Карло. Алгоритм состоит в следующем. Сначала в квадрат с центром в начале координат и со стороной два вписывается круг с единичным радиусом. Затем в этом квадрате случайным образом с равномерным распределением генерируются N точек. Точка может попасть в окружность или нет (условие попадания $x^2 + y^2 \leq 1$). Далее определяется число M точек, попавших в круг. При достаточно большом числе бросков N, по значениям M и N вычисляется число π:*

$$\pi \approx \frac{4M}{N}$$

3. *Для программы сгенерировать ассемблерные листинги для архитектуры ARM, используя различные уровни комплексной оптимизации.*
4. *Проанализировать полученные листинги и сделать следующее:*
   - *Сопоставьте 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.*
5. *Составить отчет по лабораторной работе. Отчет должен содержать следующее:*
   - *Титульный лист;*
   - *Цель лабораторной работы;*
   - *Полный компилируемый листинг реализованной программы и команды для ее компиляции;*
   - *Листинг на ассемблере с описаниями назначения команд с точки зрения реализации алгоритма выбранного варианта;*
   - *Вывод по результатам лабораторной работы.*

ОПИСАНИЕ РАБОТЫ

Пошаговое описание выполненной работы

1. *Сгенерировал ассемблерные листинги при помощи godbolt.org, используя следующие наборы флагов -O0 -march=armv7 -mfloat-abi=soft, -O3 -march=armv7 -mfloat-abi=soft, и компилятор ARM gcc 12.2 (linux);*
2. *Сопоставил команды исходного кода на Си с машинными командами, оставляя комментарии в листинге (см. Приложение 2);*

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы:

- *Познакомился с программной архитектурой ARM;*
- *Проанализировал ассемблерный листинг программы для архитектуры ARM;*
- *Сравнил архитектуры x86 и ARM;*

ПРИЛОЖЕНИЕ 1 (ЛИСТИНГ ПРОГРАММЫ НА СИ)

src / main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double MonteCarloAlgorithm(int count)
{
    double insideCount = 0.0;
    srand(time(NULL));

    for (int i = 0; i < count; ++i)
    {
        double x = (double)rand() / RAND_MAX;
        double y = (double)rand() / RAND_MAX;

        if ((x * x) + (y * y) <= 1.0)
        {
            insideCount += 4.0;
        }
    }

    return insideCount / count;
}

int main()
{
    int count = 100000000;

    double pi = MonteCarloAlgorithm(count);
    printf("PI: %lf\n", pi);

    return EXIT_SUCCESS;
}
```

ПРИЛОЖЕНИЕ 2 (АССЕМБЛЕРНЫЕ ЛИСТИНГИ ПРОГРАММЫ)

Ассемблерный листинг для архитектуры ARM и уровнем оптимизации O0

```
1.  MonteCarloAlgorithm:
2.          push    {r4, r5, r7, lr}        // поместить в стек регистры r4, r5,
3.                                          // r7, lr
4.          sub     sp, sp, #40             // выделить 40 байтов для локальных
5.                                          // переменных
6.          add     r7, sp, #0              // записать в r7 sp+0 (аналог ebp)
7.          str     r0, [r7, #4]            // записать count в стек
8.          movs    r0, #0                  // создание NULL для time
9.          bl      time                    // вызов time
10.         mov     r3, r0
11.         mov     r0, r3
12.         bl      srand                   // вызов srand с аргументом time(NULL)
13.         mov     r2, #0
14.         mov     r3, #0                  // создание insideCount (старшая и
15.                                         // младшая часть)
16.         strd    r2, [r7, #32]           // записать insideCount в стек
17.         movs    r3, #0                  // создание i
18.         str     r3, [r7, #28]           // записать i в стек
19.         b       .L2                     // перейти к метки .L2
20. .L5:
21.         bl      rand                    // вызов rand
22.         mov     r3, r0
23.         mov     r0, r3
24.         bl      __aeabi_i2d             // конвертировать rand() в double
25.         adr     r3, .L8                 // загрузить константу из .L8 (RAND_MAX)
26.                                         // в r3
27.         ldr     r2, [r3]                // загрузить RAND_MAX в r2
28.         bl      __aeabi_ddiv            // (double)rand() / RAND_MAX
29.         mov     r2, r0                  // поместить младшую часть в r2
30.         mov     r3, r1                  // поместить старшую часть в r3
31.         strd    r2, [r7, #16]           // поместить младшую и старшую часть
32.                                         // в стек (создание x)
33.         bl      rand                    // вызов rand
34.         mov     r3, r0
35.         mov     r0, r3
36.         bl      __aeabi_i2d             // конвертировать rand() в double
37.         adr     r3, .L8                 // загрузить константу из .L8 (RAND_MAX)
38.                                         // в r3
39.         ldr     r2, [r3]                // загрузить RAND_MAX в r2
40.         bl      __aeabi_ddiv            // (double)rand() / RAND_MAX
41.         mov     r2, r0                  // поместить младшую часть в r2
42.         mov     r3, r1                  // поместить старшую часть в r3
43.         strd    r2, [r7, #8]            // поместить младшую и старшую часть в
44.                                         // стек (создание y)
45.         ldrd    r2, [r7, #16]           // загрузить x в r2
46.         ldrd    r0, [r7, #16]           // загрузить x в r0
47.         bl      __aeabi_dmul            // x * x
48.         mov     r2, r0
49.         mov     r3, r1                  // сохранить результат вызова в r2 и r3
50.         mov     r4, r2
51.         mov     r5, r3                  // переписать результат вызова в r4 и r5
52.         ldrd    r2, [r7, #8]            // загрузить y в r2
53.         ldrd    r0, [r7, #8]            // загрузить y в r0
54.         bl      __aeabi_dmul            // y * y
55.         mov     r2, r0
56.         mov     r3, r1                  // сохранить результат вызова в r2 и r3
57.         mov     r0, r4
58.         mov     r1, r5                  // переписать x*x в r0 и r1
59.         bl      __aeabi_dadd            // x*x + y*y
60.         mov     r2, r0
61.         mov     r3, r1                  // сохранить результат вызова в r2 и r3
62.         mov     r0, r2
```

```
63.         mov     r1, r3                  // переписать результат вызова в r0 и r1
64.         mov     r2, #0                  // обнулить r2
65.         mov     r3, #0                  // обнулить r3
66.         movt    r3, 16368               // r3 = (double)(0 | (16368 << 16)) = 1.0
67.         bl      __aeabi_dcmple          // x*x + y*y <= 1.0 (возвращает 0, если
68.                                         // ложно, и 1, если истинно)
69.         mov     r3, r0                  // поместить результат в r3
70.         cmp     r3, #0                  // сравнить результат с 0
71.         beq     .L3                     // перейти к метке .L3, если 0
72.         mov     r2, #0                  // обнулить r2
73.         mov     r3, #0                  // обнулить r3
74.         movt    r3, 16400               // r3 = (double)(0 | (16400 << 16)) = 4.0
75.         ldrd    r0, [r7, #32]           // загрузить в r0 и r1 insideCount
76.         bl      __aeabi_dadd            // сложить insideCount и 4.0
77.         mov     r2, r0
78.         mov     r3, r1                  // переписать изменённый insideCount
79.                                         // в r2 и r3
80.         strd    r2, [r7, #32]           // поместить изменённый insideCount в стек
81. .L3:
82.         ldr     r3, [r7, #28]           // загрузить i в r3
83.         adds    r3, r3, #1              // прибавить к i 1
84.         str     r3, [r7, #28]           // загрузить изменённый i в r3
85. .L2:
86.         ldr     r2, [r7, #28]           // загрузить в r2 i
87.         ldr     r3, [r7, #4]            // загрузить в r3 count
88.         cmp     r2, r3                  // сравнить i и count
89.         blt     .L5                     // перейти к метке .L5, если i < count
90.         ldr     r0, [r7, #4]            // загрузить в r0 count
91.         bl      __aeabi_i2d             // конвернтировать count в double
92.         mov     r2, r0
93.         mov     r3, r1                  // переписать (double)count в r2 и r3
94.         ldrd    r0, [r7, #32]           // загрузить insideCount в r0 и r1
95.         bl      __aeabi_ddiv            // insideCount / count
96.         mov     r2, r0
97.         mov     r3, r1                  // переписать результат в r2 и r3
98.         mov     r0, r2
99.         mov     r1, r3                  // переписать результат в r0 и r1
100.                                        // (результат MonteCarloAlgorithm)
101.        adds    r7, r7, #40             // вернуть 40 байтов, зарезервированные
102.                                        // для локальныз переменных
103.        mov     sp, r7                  // записать r7 в sp (сбрасываем кадр
104.                                        // стека)
105.        pop     {r4, r5, r7, pc}        // вернуть изначальные значения r4,
106.                                        // r5, r7, pc
107. .L8:
108.        .word   -4194304                // RAND_MAX
109.        .word   1105199103
110. .LC0:
111.        .ascii  "PI: %lf\012\000"
112. main:
113.        push    {r7, lr}                // поместить в стек регистры r7, lr
114.        sub     sp, sp, #16             // выделить 16 байт для локальных
115.                                        // переменных
116.        add     r7, sp, #0              // записать в r7 sp+0 (аналог ebp)
117.        mov     r3, #57600              // создать count
118.        movt    r3, 1525                // r1 = 57600 | (1525 << 16) = 100000000
119.        str     r3, [r7, #12]           // записать count в стек
120.        ldr     r0, [r7, #12]           // загрузить count в r0 (аргумент функции
121.                                        // MonteCarloAlgorithm)
122.        bl      MonteCarloAlgorithm     // вызвать MonteCarloAlgorithm
123.        strd    r0, [r7]                // записать в стек результат
124.                                        // MonteCarloAlgorithm (создание pi)
125.        ldrd    r2, [r7]                // загрузить pi (второй аргумент printf)
126.        movw    r0, #:lower16:.LC0
127.        movt    r0, #:upper16:.LC0      // поместить указатель на строку из метки
128.                                        // .LC0 в r0 (первый аргумент printf)
129.        bl      printf                  // вызов printf
```

```
130.            movs    r3, #0                  // обнулить r3
131.            mov     r0, r3                  // копировать значение из r3 в r0
132.                                            // (результат main)
133.            adds    r7, r7, #16             // вернуть 16 байтов, выделенные для
134.                                            // локальных переменных
135.            mov     sp, r7                  // копировать r7 (аналог ebp) в sp
136.            pop     {r7, pc}                // вернуть изначальные значения r7, pc
```

Ассемблерный листинг для архитектуры ARM и уровнем оптимизации O3

```
1.  MonteCarloAlgorithm:
2.          push    {r4, r5, r6, r7,        // поместить в стек регистры r4, r5, r6,
3.                   r8, r9, r10, fp, lr}   // r7, r8, r9, r10, fp, lr
4.          mov     r9, r0                  // переписать count в r9
5.          movs    r0, #0                  // обнулить r0 (аргумент time)
6.          sub     sp, sp, #12             // зарезервировать 12 байтов для локальных
7.                                          // переменных
8.          movs    r4, #0                  // обнулить r4 (младшая часть insideCount)
9.          bl      time                    // вызвать time (реузьтат в r0)
10.         bl      srand                   // вызвать srand (r0 аргумент)
11.         movs    r3, #0                  // обнулить r3 (старшая часть insideCount)
12.         cmp     r9, #0                  // сравнить count и 0
13.         strd    r3, [sp]                // записать insideCount в стек
14.         ble     .L2                     // перейти к метке .L2, если count <= 0
15.         adr     r10, .L11               // загружает RAND_MAX в r10
16.         mov     r8, #0                  // обнулить r8 (создание i)
17. .L5:
18.         bl      rand                    // вызвать rand
19.         add     r8, r8, #1              // ++i
20.         bl      __aeabi_i2d             // конвертировать результат rand()
21.                                         // в double
22.         mov     r2, r10                 // переписать RAND_MAX в r2
23.         bl      __aeabi_ddiv            // (double)rand() / RAND_MAX (создание x)
24.         mov     r5, r1
25.         mov     r4, r0                  // переписать x в r4 и r5
26.         bl      rand                    // вызвать rand
27.         bl      __aeabi_i2d             // конвертировать результат rand() в
28.                                         // double
29.         mov     r2, r10                 // переписать RAND_MAX в r2
30.         bl      __aeabi_ddiv            // (double)rand() / RAND_MAX (создание y)
31.         mov     r6, r0
32.         mov     r7, r1                  // переписать y в r6 и r7
33.         mov     r2, r4
34.         mov     r3, r5                  // переписать x в r2 и r3
35.         mov     r0, r4
36.         mov     r1, r5                  // переписать x в r0 и r1
37.         bl      __aeabi_dmul            // x*x
38.         mov     r4, r0
39.         mov     r5, r1                  // переписать x*x в r4 и r5
40.         mov     r2, r6
41.         mov     r3, r7                  // переписать y в r2 и r3
42.         mov     r0, r6
43.         mov     r1, r7                  // переписать y в r0 и r1
44.         bl      __aeabi_dmul            // y*y
45.         mov     r2, r0
46.         mov     r3, r1                  // переписать x*x в r2 и r3
47.         mov     r0, r4
48.         mov     r1, r5                  // переписать x*x в r0 и r1
49.         bl      __aeabi_dadd            // y*y + x*x
50.         movs    r2, #0                  // обнулить r2
51.         movs    r3, #0                  // обнулить r3
52.         movt    r3, 16368               // 0 | (16368 << 16) = 1.0 (в r2 и r3)
53.         bl      __aeabi_dcmple          // x*x + y*y <= 1.0 (возвращает 0, если
54.                                         // ложно, и 1, если истинно)
55.         movs    r3, #0                  // обнулить r3
56.         movt    r3, 16400               // 0 | (16400 << 16) = 4.0 (в r2 и r3)
```

```
57.         mov     r4, r0                  // переписать результат x*x + y*y <= 1.0
58.                                         // в r4
59.         movs    r2, #0                  // обнулить r2
60.         ldrd    r0, [sp]                // загрузить insideCount в r0, r1
61.         cbz     r4, .L3                 // перейти к метке .L3, если результат x*x
62.                                         // + y*y <= 1.0 равен 0
63.         bl      __aeabi_dadd            // insideCount + 4.0
64.         strd    r0, [sp]                // записать insideCount + 4.0 в стек
65.  .L3:
66.         cmp     r9, r8                  // сравнить count и i
67.         bne     .L5                     // перейти к метке .L5, если count != i
68.  .L2:
69.         mov     r0, r9                  // переписать count в r0
70.         bl      __aeabi_i2d             // конвертировать count в double
71.         mov     r2, r0
72.         mov     r3, r1                  // переписать (double)count в r2 и r3
73.         ldrd    r0, [sp]                // загрузить insideCount в r0, r1
74.         bl      __aeabi_ddiv            // insideCount / count
75.         add     sp, sp, #12             // вернуть 12 байтов, зарезервированные
76.                                         // для локальныз переменных
77.         pop     {r4, r5, r6, r7,        // вернуть изначальные значения r4, r5,r6,
78.                  r8, r9, r10, fp, pc}   // r7, r8, r9, r10, fp, lr
79.  .L11:
80.         .word   -4194304                // RAND_MAX
81.         .word   1105199103
82.  .LC0:
83.         .ascii  "PI: %lf\012\000"
84.  main:
85.         push    {r3, lr}                // поместить в стек регистры r3, lr
86.         mov     r0, #57600
87.         movt    r0, 1525                // r0 = 57600 | (1525 << 16) = 1.0
88.         bl      MonteCarloAlgorithm     // вызвать MonteCarloAlgorithm
89.         mov     r2, r0
90.         mov     r3, r1                  // переписать pi в r2, r3 (второй аргумент
91.                                         // printf)
92.         movw    r0, #:lower16:.LC0
93.         movt    r0, #:upper16:.LC0      // поместить указатель на строку из метки
94.                                         // .LC0 в r0 (первый аргумент printf)
95.         bl      printf                  // вызвать MonteCarloAlgorithm
96.         movs    r0, #0                  // обнулить r0 (результат main)
97.         pop     {r3, pc}                // вернуть изначальные значения r3, pc
```