###### **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**

**Практическая работа №4**

ВВЕДЕНИЕ В АРХИТЕКТУРУ ARM

студента 2 курса, группы 23201

Сорокина Матвея Павловича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.С. Матвеев

Новосибирск 2024

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ** 3](#_Toc175850303)

[**ЗАДАНИЕ** 3](#_Toc175850304)

[**ОПИСАНИЕ РАБОТЫ** 4](#_Toc175850305)

[**ЗАКЛЮЧЕНИЕ** 8](#_Toc175850306)

[**ПРИЛОЖЕНИЯ** 9](#_Toc175850307)

[**Приложение 1:Исходный код программы *LogCalculation.cpp*** 9](#_Toc175850308)

[**Приложение 2: Разбивка ассемблерного кода с объяснением размещения переменных функции LogCalculation(long double x, long long n)** 10](#_Toc175850309)

[**Приложение 3: ассемблерный листинг программы LogCalculation.cpp с ключом контроля оптимизации -O0, архитектурой ARM.** 11](#_Toc175850310)

# **ЦЕЛЬ**

Знакомство с программной архитектурой ARM и анализ ассемблерного листинга программы для архитектуры ARM.

# **ЗАДАНИЕ**

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 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.

1. Составить отчет, отражающий этапы работы, результаты анализа, выводы.

# **ОПИСАНИЕ РАБОТЫ**

1. Реализовано задание 6 (из лабораторной работы 1) - алгоритм вычисления функции *ln(1+x)* с помощью разложения в степенной ряд по первым N членам данного ряда на языке *C++.* Код программы предоставлен (см. Приложение 1).

Для измерения времени работы программы использовалась библиотечная функция *clock\_gettime* из библиотеки *time.h*. Замеры времени были произведены до начала и после завершения работы функции, вычисляющей *ln(1+x)*.

Разность этих двух значений дает общее время выполнения функции. Для проверки точности измерений, код программы запускается несколько раз.

1. Ассемблерные листы с различными уровнями оптимизации генерировались с помощью:

* Компилятора:

Так как изначально не был установлен компилятор для ARM, использовался инструмент ***arm-none-eabi-gcc***, который является частью *ARM GNU Toolchain*.

Команды для компиляции файла *LogCalculation.cpp* и получения его ассемблерного листинга:

|  |
| --- |
| arm-none-eabi-g++ -S -O0 -o res\_O0.s LogCalculation.cpp |
| arm-none-eabi-g++ -S -O3 -o res\_O0.s LogCalculation.cpp |

Флаги ***-march*** генерируют ассемблерные коды, оптимизированные для конкретной архитектуры:

|  |
| --- |
| arm-none-eabi-g++ -S -march=armv7-a -o res\_armv7.s LogCalculation.cpp |
| arm-none-eabi-g++ -S -march=armv8-a -o res\_armv8.s LogCalculation.cpp |

***-march=armv7-a***: флаг для компиляции с оптимизацией для архитектуры ARMv7 (32-битной).

***-march=armv8-a:*** флаг для компиляции с оптимизацией для архитектуры ARMv8 (64-битной).

* Сайта: <https://godbolt.org>

1. Сопоставление команд языка Си с машинными командами, размещение переменных языка Си в программах на ассемблере на примере функции *long double LogCalculation(long double x, long long n).*

Функция на языке *C++:*

|  |
| --- |
| long double LogCalculation(long double x, long long n) {      long double res = 0;      long double prev = x;      long double sign = 1.0;      for (long long i = 1; i <= n; i++) {          res += sign \* prev / i;          prev \*= x;          sign = -sign;      }      return res;  } |

Разбивка ассемблерного кода с пояснениями размещения переменных предоставлена (см. Приложение 2).

Когда мы пишем функцию на *С*++, значения ее параметров переходят в регистры или в стек вызовов в зависимости от соглашения о вызовах для компилятора. Соглашения о вызовах (Calling convention) определяет, как параметры передаются в функции и как результат возвращается. Для *ARM*:

* Параметры функций обычно передаются в регистрах: ***r0 – r3*** для целых чисел и ***d0 – d1*** для чисел с плавающей точкой.
* Локальные переменные хранятся в памяти (стеке вызовов) или в регистрах.

На ассемблере при вызове функции инициализируются некоторые переменные, которые могу хранится как в регистрах, так и в стеке, например:

|  |
| --- |
| vstr.64 d0, [r7, #8] ; x сохраняется по адресу [r7 + 8] |

* инструкция ***vstr.64 d0, [r7, #8]***: значение *x* (передается в регистре d0) сохраняется по адресу *[r7+8]* в стеке. Важно также понимать, что r7 – указатель на начало стек, #8 – смещение в байтах.
* инструкция ***vldr.64 d17, [r7, #40]***: значение *res*, сохраненное в стеке по адресу *[r7 + 40]*, загружается в регистр *d17.* Это делается для использования переменной *res* в арифметических операциях позже.

Мы видим, что переменные дублируются в памяти компьютера. У этого есть несколько причин:

* + 1. **Локальные переменные:** после записи параметров функции в регистры, для сохранения промежуточных значений и локальных переменных выделяется место в стеке. Пример –переменная *res* в функции *LogCalculation(long double x, long long n):*

|  |
| --- |
| .L3:  …  strd r2, [r7, #40] ; res инициализируется и сохраняется по адресу [r7 + 40]  …  vdiv.f64 d16, d8, d17 ; деление (sign \* prev) / i  vldr.64 d17, [r7, #40] ; загрузка res из стека  vadd.f64 d16, d17, d16 ; res += (sign \* prev) / i  … |

* + 1. **Сохранение состояния:** регистры используются для промежуточных вычислений, их значения могут быть сохранены в стеке, для избежания потери данных при изменении регистра.

|  |
| --- |
| .L3:  …  vldr.64 d17, [r7, #24] ; загрузка sign из стека в регистр d17  vldr.64 d16, [r7, #32] ; загрузка prev из стека в регистр d16  vmul.f64 d8, d17, d16 ; умножение sign на prev  …  vdiv.f64 d16, d8, d17 ; деление (sign \* prev) / i  … |

1. Используя Приложение 2 ниже представлено размещение переменных в ассемблере:

Входные параметры:

* *long double x*: регистр *d0*; адрес в стеке *[r7+8];*
* *long double n*: регистр *r0*; вдрес в стеке *[r7];*

Локальные переменные:

* *long double res*: инициализируется в 0; адрес в стеке *[r7+40];*
* *long double prev*: инициализируется как *x*; адрес в стеке *[r7+32];*
* *long double sign*: изначально 1.0; адрес в стеке *[r7+24];*
* *long double i*: изначально 1: адрес в стеке *[r7+16].*

1. Использование ключевых особенностей архитектуры *ARM* на конкретных участках ассемблерного кода.
   * + - **Использование регистров для передачи параметров и возврата значений**

В *ARM*-процессорах используется набор регистров общего назначения (r0-r15) и специальные регистры для чисел с плавающей точкой (d0-d31).

Передача параметров в функцию часто осуществляется через регистры, а не через стек, что ускоряет вызов функции.

Пример из функции *LogCalculation(long double x, long long n):*

|  |
| --- |
| vstr.64 d0, [r7, #8] ; сохраняет первый параметр (x) в стеке по адресу [r7+8] |

В примере x (первый параметр функции) передается через регистр d0 и сохраняется в стеке для последующего использования.

* + - * **Инструкции для работы с числами с плавающей точкой**

Инструкции ***vadd.f64, vmul.f64***, ***vdiv.f64***, позволяют нам выполнять арифметические операции с числами с плавающей точкой.

Именно суффикс ***.f64*** указывает на то, что эти операции выполняются с 64-битными числами с плавающей точкой (double в *C++*).

Пример:

|  |
| --- |
| vmul.f64 d8, d17, d16 ; умножение двух значений с плавающей точкой |

В примере два значения с плавающей точкой, хранящиеся в регистрах *d17* и *d16*, и сохраняет результат в регистре *d8*.

* + - * **Инструкции управления стеком**

*ARM* использует стек для сохранения контекста при вызове функций, примером являются инструкции **push** и **pop**.

Пример:

|  |
| --- |
| push {r4, r5, r7, lr} ; сохраняет регистры и адрес возврата в стеке |
| pop {r4, r5, r7, pc} ; восстановление регистров и возврат из функции |

Использование стека вызовов в ARM делится на две основные категории:

**Сохранение контекста**: ***push*** и ***pop***, для сохранения и восстановления регистров при вызове функций.

**Работа с переменными**: ***vstr***, ***vldr***, для временного сохранения переменных в стеке.

# **ЗАКЛЮЧЕНИЕ**

В ходе данной лабораторной работы я познакомился с программной архитектурой ARM и научился анализировать ассемблерный листинг программы для архитектуры ARM.

# **ПРИЛОЖЕНИЯ**

# **Приложение 1:Исходный код программы *LogCalculation.cpp***

|  |
| --- |
| #include <iostream>  #include <time.h>  #include <cstdlib> // for atoi and atof  long double LogCalculation(long double x, long long n) {      long double res = 0;      long double prev = x;      long double sign = 1.0;      for (long long i = 1; i <= n; i++) {          res += sign \* prev / i;          prev \*= x;          sign = -sign;      }      return res;  }  int main(int argc, char \*argv[]) {      if (argc != 3) {          std::cerr << "Usage: <base> <number of terms>" << std::endl;          return 0;      }      struct timespec start, end;      long double x = atoll(argv[1]);      if (x <= -1 || x > 1) {          std::cerr << "x must be in the range (-1, 1]" << std::endl;      }      long long n = atoll(argv[2]);      std::cout << "x = " << x << ", n = " << n << std::endl;      int runs = 5;      double time\_total = 0;      for (long long i = 0; i < runs; i++) {          clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);          long double log = LogCalculation(x, n);          clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);          double taken\_time = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;          std::cout << "log(" << x << ") = " << log << std::endl;          std::cout << "Run #" << i + 1 << " took " << taken\_time << " seconds to complete" << "\n" << std::endl;          time\_total += taken\_time;      }      std::cout << "Average time: " << time\_total / runs << " seconds" << std::endl;      return 0;  } |

# **Приложение 2: Разбивка ассемблерного кода с объяснением размещения переменных функции LogCalculation(long double x, long long n)**

|  |
| --- |
| LogCalculation(long double, long long):  push {r4, r5, r7, lr} ; cохранение регистров, которыми будем пользоваться  vpush.64 {d8} ; Сохранение регистра с плавающей точкой  sub sp, sp, #48 ; Выделение 48 байт в стеке для локальных переменных – для переменных res, prev, sign и i соответственно  add r7, sp, #0 ; установка указателя на стек  ; Размещение переменных в локальном стеке (стеке вызовов)  vstr.64 d0, [r7, #8] **; x (добвляемое в d0) сохраняется по адресу [r7 + 8] в стеке вызовов**  strd r0, [r7] ; **n (r0) сохраняется по адресу [r7]**  mov r2, #0 ; res (инициализируемое/начальное значение: 0)  mov r3, #0 ; prev (ничальное значение: 0)  strd r2, [r7, #40] ; **res (d0) сохраняется по адресу [r7 + 40]**  ldrd r2, [r7, #8] ; Загрузка x из стека  strd r2, [r7, #32] ; **prev (d2) сохраняется по адресу [r7 + 32]**  mov r2, #0 ; инициализация переменной sign (1.0)  mov r3, #0  movt r3, 16368 ; sign (инициализируется значением 1.0)  strd r2, [r7, #24] ; **sign сохраняется по адресу [r7 + 24]**  mov r2, #1  mov r3, #0  strd r2, [r7, #16] **; i (инициализируется значением 1), сохраняется по адресу [r7 + 16]**  b .L2 ; Переход к циклу  .L3:  vldr.64 d17, [r7, #24] ; загрузка sign в регистр d17 для итерации цикла в функции из стека по адресу [r7+24]  vldr.64 d16, [r7, #32] ; загрузка prev в d16  vmul.f64 d8, d17, d16 ; **d8 = sign \* prev**  ldrd r0, [r7, #16] ; загрузка i в r0  bl \_\_aeabi\_l2d ; Преобразование i в тип double  vmov d17, r0, r1 ; В d17 загружаем результат  vdiv.f64 d16, d8, d17 ; **d16 = (sign \* prev) / i**  vldr.64 d17, [r7, #40] ; загрузка res в регистр d17  vadd.f64 d16, d17, d16 ; res += (sign \* prev) / i  vstr.64 d16, [r7, #40] ; сохранение результата res из регистра d16 в память  vldr.64 d17, [r7, #32] ; Загрузка prev (в d17)  vldr.64 d16, [r7, #8] ; Загрузка x (в d16)  vmul.f64 d16, d17, d16 ; **prev \*= x**  vstr.64 d16, [r7, #32] ; Сохранение обновленного prev  vldr.64 d16, [r7, #24] ; Загрузка sign (в d16)  vneg.f64 d16, d16 ; **sign = -sign**  vstr.64 d16, [r7, #24] ; Сохранение нового sign из d16 в память  ldrd r2, [r7, #16] ; Загрузка i (r2)  adds r4, r2, #1 ; **i++**  adc r5, r3, #0 ; Обновление r5 для i++  strd r4, [r7, #16] ; Сохранение нового i в r4  .L2:  ldrd r2, [r7, #16] ; загрузка i (r2)  ldrd r0, [r7] ; загрузка n (r0)  cmp r0, r2 ; сравнение i и n (регистры: r0 и r2)  sbcs r3, r1, r3 ; обновление флагов  bge .L3 ; переход к циклу .L3 (выше), **если** i <= n  ldrd r2, [r7, #40] ; загрузка res (r2)  vmov d16, r2, r3 ; в d16 загружается res  vmov.f64 d0, d16 ; возвращение результата функции res (из регистра d16) в регистр d0  adds r7, r7, #48  mov sp, r7  vldm sp!, {d8}  pop {r4, r5, r7, pc} ; восстановление регистров и возврат из функции (извлекаются из стека) ; восстановление pc (счетчика команд/program counter) из стека значит, что выполнение программы вернется к адресу, который был сохранен ранее, туда, откуда была вызвана функция. |

# **Приложение 3: Ассемблерный листинг программы LogCalculation.cpp с ключом контроля оптимизации -O0, архитектурой ARM.**

|  |
| --- |
| LogCalculation(long double, long long):          push    {r4, r5, r7, lr}          vpush.64        {d8}          sub     sp, sp, #48          add     r7, sp, #0          vstr.64 d0, [r7, #8]          strd    r0, [r7]          mov     r2, #0          mov     r3, #0          strd    r2, [r7, #40]          ldrd    r2, [r7, #8]          strd    r2, [r7, #32]          mov     r2, #0          mov     r3, #0          movt    r3, 16368          strd    r2, [r7, #24]          mov     r2, #1          mov     r3, #0          strd    r2, [r7, #16]          b       .L2  .L3:          vldr.64 d17, [r7, #24]          vldr.64 d16, [r7, #32]          vmul.f64        d8, d17, d16          ldrd    r0, [r7, #16]          bl      \_\_aeabi\_l2d          vmov    d17, r0, r1          vdiv.f64        d16, d8, d17          vldr.64 d17, [r7, #40]          vadd.f64        d16, d17, d16          vstr.64 d16, [r7, #40]          vldr.64 d17, [r7, #32]          vldr.64 d16, [r7, #8]          vmul.f64        d16, d17, d16          vstr.64 d16, [r7, #32]          vldr.64 d16, [r7, #24]          vneg.f64        d16, d16          vstr.64 d16, [r7, #24]          ldrd    r2, [r7, #16]          adds    r4, r2, #1          adc     r5, r3, #0          strd    r4, [r7, #16]  .L2:          ldrd    r2, [r7, #16]          ldrd    r0, [r7]          cmp     r0, r2          sbcs    r3, r1, r3          bge     .L3          ldrd    r2, [r7, #40]          vmov    d16, r2, r3          vmov.f64        d0, d16          adds    r7, r7, #48          mov     sp, r7          vldm    sp!, {d8}          pop     {r4, r5, r7, pc}  .LC0:          .ascii  "Usage: <base> <number of terms>\000"  .LC1:          .ascii  "x must be in the range (-1, 1]\000"  .LC2:          .ascii  "x = \000"  .LC3:          .ascii  ", n = \000"  .LC4:          .ascii  "log(\000"  .LC5:          .ascii  ") = \000"  .LC6:          .ascii  "Run #\000"  .LC7:          .ascii  " took \000"  .LC8:          .ascii  " seconds to complete\000"  .LC9:          .ascii  "\012\000"  .LC10:          .ascii  "Average time: \000"  .LC11:          .ascii  " seconds\000"  main:          push    {r4, r5, r7, r8, r9, r10, fp, lr}          sub     sp, sp, #80          add     r7, sp, #0          str     r0, [r7, #4]          str     r1, [r7]          ldr     r3, [r7, #4]          cmp     r3, #3          beq     .L6          movw    r1, #:lower16:.LC0          movt    r1, #:upper16:.LC0          movw    r0, #:lower16:std::cerr          movt    r0, #:upper16:std::cerr          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))          movs    r3, #0          b       .L13  .L6:          ldr     r3, [r7]          adds    r3, r3, #4          ldr     r3, [r3]          mov     r0, r3          bl      atoll          mov     r2, r0          mov     r3, r1          mov     r0, r2          mov     r1, r3          bl      \_\_aeabi\_l2d          mov     r2, r0          mov     r3, r1          strd    r2, [r7, #56]          vldr.64 d16, [r7, #56]          vmov.f64        d17, #-1.0e+0          vcmpe.f64       d16, d17          vmrs    APSR\_nzcv, FPSCR          bls     .L8          vldr.64 d16, [r7, #56]          vmov.f64        d17, #1.0e+0          vcmpe.f64       d16, d17          vmrs    APSR\_nzcv, FPSCR          ble     .L9  .L8:          movw    r1, #:lower16:.LC1          movt    r1, #:upper16:.LC1          movw    r0, #:lower16:std::cerr          movt    r0, #:upper16:std::cerr          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))  .L9:          ldr     r3, [r7]          adds    r3, r3, #8          ldr     r3, [r3]          mov     r0, r3          bl      atoll          strd    r0, [r7, #48]          movw    r1, #:lower16:.LC2          movt    r1, #:upper16:.LC2          movw    r0, #:lower16:std::cout          movt    r0, #:upper16:std::cout          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          vldr.64 d0, [r7, #56]          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long double)          mov     r3, r0          movw    r1, #:lower16:.LC3          movt    r1, #:upper16:.LC3          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r1, r0          ldrd    r2, [r7, #48]          mov     r0, r1          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long long)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))          movs    r3, #5          str     r3, [r7, #44]          mov     r2, #0          mov     r3, #0          strd    r2, [r7, #72]          vmov.i32        d16, #0  @ di          vstr.64 d16, [r7, #64]    @ int          b       .L11  .L12:          add     r3, r7, #16          mov     r1, r3          movs    r0, #4          bl      clock\_gettime          ldrd    r0, [r7, #48]          vldr.64 d0, [r7, #56]          bl      LogCalculation(long double, long long)          vstr.64 d0, [r7, #32]          add     r3, r7, #8          mov     r1, r3          movs    r0, #4          bl      clock\_gettime          ldr     r2, [r7, #8]          ldr     r3, [r7, #16]          subs    r3, r2, r3          vmov    s15, r3 @ int          vcvt.f64.s32    d17, s15          ldr     r2, [r7, #12]          ldr     r3, [r7, #20]          subs    r3, r2, r3          vmov    s15, r3 @ int          vcvt.f64.s32    d18, s15          vldr.64 d19, .L14          vdiv.f64        d16, d18, d19          vadd.f64        d16, d17, d16          vstr.64 d16, [r7, #24]          movw    r1, #:lower16:.LC4          movt    r1, #:upper16:.LC4          movw    r0, #:lower16:std::cout          movt    r0, #:upper16:std::cout          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          vldr.64 d0, [r7, #56]          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long double)          mov     r3, r0          movw    r1, #:lower16:.LC5          movt    r1, #:upper16:.LC5          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          vldr.64 d0, [r7, #32]          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long double)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))          movw    r1, #:lower16:.LC6          movt    r1, #:upper16:.LC6          movw    r0, #:lower16:std::cout          movt    r0, #:upper16:std::cout          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r1, r0          ldrd    r2, [r7, #64]          adds    r10, r2, #1          adc     fp, r3, #0          mov     r2, r10          mov     r3, fp          mov     r0, r1          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long long)          mov     r3, r0          movw    r1, #:lower16:.LC7          movt    r1, #:upper16:.LC7          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          vldr.64 d0, [r7, #24]          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(double)          mov     r3, r0          movw    r1, #:lower16:.LC8          movt    r1, #:upper16:.LC8          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          movw    r1, #:lower16:.LC9          movt    r1, #:upper16:.LC9          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))          vldr.64 d17, [r7, #72]          vldr.64 d16, [r7, #24]          vadd.f64        d16, d17, d16          vstr.64 d16, [r7, #72]          ldrd    r2, [r7, #64]          adds    r8, r2, #1          adc     r9, r3, #0          strd    r8, [r7, #64]  .L11:          ldr     r3, [r7, #44]          asrs    r2, r3, #31          mov     r4, r3          mov     r5, r2          ldrd    r2, [r7, #64]          cmp     r2, r4          sbcs    r3, r3, r5          blt     .L12          movw    r1, #:lower16:.LC10          movt    r1, #:upper16:.LC10          movw    r0, #:lower16:std::cout          movt    r0, #:upper16:std::cout          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          ldr     r2, [r7, #44]          vmov    s15, r2 @ int          vcvt.f64.s32    d16, s15          vldr.64 d17, [r7, #72]          vdiv.f64        d18, d17, d16          vmov.f64        d0, d18          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(double)          mov     r3, r0          movw    r1, #:lower16:.LC11          movt    r1, #:upper16:.LC11          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*)          mov     r3, r0          movw    r1, #:lower16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          movt    r1, #:upper16:std::basic\_ostream<char, std::char\_traits<char> >& std::endl<char, std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&)          mov     r0, r3          bl      std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&))          movs    r3, #0  .L13:          mov     r0, r3          adds    r7, r7, #80          mov     sp, r7          pop     {r4, r5, r7, r8, r9, r10, fp, pc}  .L14:          .word   0          .word   1104006501 |