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 архитектуру x86-64:»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2022

СОДЕРЖАНИЕ

Цель

1. Знакомство с программной архитектурой x86-64.

2. Анализ ассемблерного листинга программы для архитектуры x86-64.

Задание

1. Изучить программную архитектуру x86-64:

· набор регистров,

· основные арифметико-логические команды,

· способы адресации памяти,

· способы передачи управления,

· работу со стеком,

· вызов подпрограмм,

· передачу параметров в подпрограммы и возврат результатов,

· работу с арифметическим сопроцессором,

· работу с векторными расширениями.

2. Для программы на языке Си++ (из практической работы 1) сгенерировать ассемблерные листинги (синтаксис AT&T, принятый в UNIX) для архитектуры архитектуры x86-64, 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);

· выписать оптимизационные преобразования, выполненные компилятором;

· (опционально) сравнить различия в программах для архитектуры x86 и архитектуры x86-64.

4. Составить отчет по лабораторной работе. Отчет должен содержать следующее:

· Титульный лист.

· Цель лабораторной работы.

· Полный компилируемый листинг реализованной программы на Си.

· Листинги на ассемблере (O0 и O3).

· Подробное описание найденных оптимизаций, примененных на уровне O3.

· Вывод по результатам лабораторной работы.

Описание работы

1. Код программы на C++ задачи про алгоритм вычисления функции ln(1+x) с помощью разложения в ряд по первым N членам этого ряда был переведен на ассемблер с помощью сайта godbolt.org. Компиляция была выполнена под архитектуру x86-64 с ключами оптимизации -O0 и -O3

2. Далее был проведен анализ полученных кодов - были добавлены комментарии к ассемблерным листингам (см. Приложения)

3. Были выявлены отличия ассемблерных листингов с разными ключами оптимизации (см. Заключение)

Формулировка задания:

Алгоритм вычисления функции ln(1+x) с помощью разложения в ряд по первым N членам этого ряда:

$$ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n+1} \frac{x^n}{n} + \ldots$$

Область сходимости ряда: - 1 ≤ x ≤1 .

Заключение

При выполнении работы была изучена программная архитектура x86-64 (было сгенерировано два файла с разными ключами оптимизации).

Оптимизационные преобразования, выполненные компилятором:

1. При ключе оптимизации -O3 была выполнена такая оптимизация, как инлайнинг: в main не осуществлялся переход к блоку функции, команды функции находились в самом main (но при этом сама функция также была переведена в ассемблер)
2. atof и atoll были заменены соответственно на strtod и strtoll, так как strtod и strtoll более стабильны в своей работе
3. При оптимизации с -O0 выделяется 80 байта для хранения локальных переменных, а при -O3 - 48 байта
4. При оптимизации -O3 сразу идет проверка 0 < n - если условие не выполняется, то цикл не начинается
5. При оптимизации -O3 уменьшено количество обращений к стеку, компилятор отображает данные на регистр

Приложение 1.1 Полный листинг программы.

```
1.  #include <iostream>
2.  #include <cstdlib>
3.  #include <exception>
4.  #include <iomanip>
5.  #include <ctime>
6.
7.  using namespace std;
8.
9.  double CountLogarithm(double x, long long int n)
10. {
11.     double result = 0;
12.
13.     for (long long int i = 1; i <= n; i++)
14.     {
15.         if (i % 2 == 1)
16.         {
17.             result += x / i;
18.         }
19.         else
20.         {
21.             result -= x / i;
22.         }
23.         x *= x;
24.     }
25.
26.     return result;
27. }
28.
29. int main(int argc, char *argv[])
30. {
31.     struct timespec start, end;
32.     double totalTime = 0;
33.
34.     double x = atof(argv[1]);
35.     long long int n = atoll(argv[2]);
```

```
36.     clock_gettime(CLOCK_MONOTONIC_RAW, &start);
37.     cout << CountLogarithm(x, n) << endl;
38.     clock_gettime(CLOCK_MONOTONIC_RAW, &end);
39.
40.     double time = end.tv_sec - start.tv_sec + 1e-9 * (end.tv_nsec - start.tv_nsec);
41.     cout << time << endl;
42.
43.     return EXIT_SUCCESS;
44. }
```

Приложение 1.2 Полный листинг программы на ассемблере x86-64 -O0

```
1.  CountLogarithm(double, long long):
2.          pushq   %rbp                    // Указатель базы кадра стека помещается в
    стек
3.          movq    %rsp, %rbp              // В регистр записывается указатель
    верхушки стека
4.          movsd   %xmm0, -24(%rbp)
5.          movq    %rdi, -32(%rbp)
6.          pxor    %xmm0, %xmm0            // Обнуление %xmm0 создание result = 0
7.          movsd   %xmm0, -8(%rbp)         // Запись значения регистра в ОЗУ
8.          movq    $1, -16(%rbp)           // Запись i = 1 в ОЗУ
9.          jmp     .L2
10. .L5:
11.         movq    -16(%rbp), %rdx         // Начало итерации цикла
12.         movq    %rdx, %rax              ////////////////////////////////////////////////////////////////
13.         sarq    $63, %rax               //                                                            //
14.         shrq    $63, %rax               //   Вычисление остатка от деление на 2                       //
15.         addq    %rax, %rdx              //                                                            //
16.         andl    $1, %edx                //                                                            //
17.         subq    %rax, %rdx              //                                                            //
18.         movq    %rdx, %rax              ////////////////////////////////////////////////////////////////
19.         cmpq    $1, %rax                // сравнение остатка от деления с единицей
20.         jne     .L3
21.         pxor    %xmm1, %xmm1            ////////////////////////////////////////////////////////////////
22.         cvtsi2sdq       -16(%rbp), %xmm1    //        Выполнение операции                 //
23.         movsd   -24(%rbp), %xmm0        //        x / i                               //
24.         divsd   %xmm1, %xmm0            ////////////////////////////////////////////////////////////////
25.         movsd   -8(%rbp), %xmm1         ////////////////////////////////////////////////////////////////
26.         addsd   %xmm1, %xmm0            //       Выполнение операции                  //
27.         movsd   %xmm0, -8(%rbp)         //       result +=                            //
28.         jmp     .L4                     ////////////////////////////////////////////////////////////////
29. .L3:
30.         pxor    %xmm2, %xmm2            ////////////////////////////////////////////////////////////////
31.         cvtsi2sdq       -16(%rbp), %xmm2    //                                                        //
32.         movsd   -24(%rbp), %xmm0        //                      x / i                             //
33.         movapd  %xmm0, %xmm1            //                                                        //
```

```
34.     divsd   %xmm2, %xmm1            ////////////////////////////////////////////////////////////////
35.     movsd   -8(%rbp), %xmm0         ////////////////////////////////////////////////////////////////
36.     subsd   %xmm1, %xmm0            //          result -=                                         //
37.     movsd   %xmm0, -8(%rbp)         ////////////////////////////////////////////////////////////////
38. .L4:
39.     movsd   -24(%rbp), %xmm0        ////////////////////////////////////////////////////////////////
40.     mulsd   %xmm0, %xmm0            //                    x *= x                                 //
41.     movsd   %xmm0, -24(%rbp)        ////////////////////////////////////////////////////////////////
42.     addq    $1, -16(%rbp)
43. .L2:
44.     movq    -16(%rbp), %rax         // Запись значения счётчика на регистр
45.     cmpq    -32(%rbp), %rax         // Сравнение i  и n
46.     jle     .L5
47.     movsd   -8(%rbp), %xmm0         // return, запись результата на регистр

48.     movq    %xmm0, %rax             // result
49.     movq    %rax, %xmm0



50.     popq    %rbp                    // Удаление элемента из стека
51.     ret
52. main:
53.     pushq   %rbp                    // Значение регистра помещается стек
54.     movq    %rsp, %rbp
55.     subq    $80, %rsp               // Сдвиг верхушки стека для локальных
    переменных
56.     movl    %edi, -68(%rbp)         // Копирование argc в ОЗУ
57.     movq    %rsi, -80(%rbp)         // Копирование argv[][] в ОЗУ
58.     pxor    %xmm0, %xmm0            // Обнуление регистра для totalTime
59.     movsd   %xmm0, -8(%rbp)         // Запись totalTime в регистр
60.     movq    -80(%rbp), %rax         // Взятие первого аргумента строки
61.     addq    $8, %rax                // Сдвиг на 8 байт, чтобы %rax стал
    указывать на argv[1]
62.     movq    (%rax), %rax            // Запись значения argv[1] в регистр
63.     movq    %rax, %rdi              ////////////////////////////////////////////////////////////////
```

```
64.     call    atof                    //              atof                //
65.     movq    %xmm0, %rax             //                                  //
66.     movq    %rax, -16(%rbp)         ////////////////////////////////////////////////////////////////
67.     movq    -80(%rbp), %rax         // Взятие второго аргумента
68.     addq    $16, %rax               // Сдвиг на 16 байт, чтобы %rax стал
    указывать на argv[1]
69.     movq    (%rax), %rax            // Запись в регистр значения по этому адресу
70.     movq    %rax, %rdi              // Запись параметра в регистр функции atoll
71.     call    atoll                   // Выполнение atoll
72.     movq    %rax, -24(%rbp)         // Запись результата функции в ОЗУ
73.     leaq    -48(%rbp), %rax         ////////////////////////////////////////////////////////////////
74.     movq    %rax, %rsi              // clock_gettime(CLOCK_MONOTONIC_   //
75.     movl    $4, %edi                //  RAW, &start);                   //
76.     call    clock_gettime           ////////////////////////////////////////////////////////////////
77.     movq    -24(%rbp), %rdx
78.     movq    -16(%rbp), %rax
79.     movq    %rdx, %rdi              // Запись параметра функции в регистр
80.     movq    %rax, %xmm0             // Запись параметра функции в регистр
81.     call    CountLogarithm(double, long long)
82.     movq    %xmm0, %rax             // Запись результата функции в регистр
83.     movq    %rax, %xmm0
84.     movl    $_ZSt4cout, %edi
85.     call    std::basic_ostream<char, std::char_traits<char> >::operator<<(double)
86.     movl    $_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_, %esi
87.     movq    %rax, %rdi
88.                     call        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> >&))
89.     leaq    -64(%rbp), %rax
90.     movq    %rax, %rsi
91.     movl    $4, %edi
92.     call    clock_gettime
93.     movq    -64(%rbp), %rdx         // Запись значения в регистр

94.     movq    -48(%rbp), %rax
```

```
95.   subq    %rax, %rdx                  // Вычисление разницы показаний таймера
96.       pxor    %xmm1, %xmm1

97.       cvtsi2sdq      %rdx, %xmm1
98.       movq    -56(%rbp), %rdx              // Взятие по полю tv_sec для start
99.       movq    -40(%rbp), %rax              // Взятие по полю tv_sec для end

100.          subq    %rax, %rdx
101.          pxor    %xmm2, %xmm2             ////////////////////////////////////////////////////////////////
102.          cvtsi2sdq      %rdx, %xmm2       //   Умножение результата разности на      //
103.          movsd   .LC1(%rip), %xmm0        //   1e-9                                  //
104.          mulsd   %xmm2, %xmm0             ////////////////////////////////////////////////////////////////
105.          addsd   %xmm1, %xmm0             // Присваивание значение
106.          movsd   %xmm0, -32(%rbp)         // переменной double totalTimet
107.          movq    -32(%rbp), %rax          // Строки 107 - 110 вывод time на экран
108.          movq    %rax, %xmm0
109.          movl    $_ZSt4cout, %edi
110.          call    std::basic_ostream<char, std::char_traits<char> >::operator<<(double)
111.          movl    $_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_, %esi
112.          movq    %rax, %rdi
113.                                  call                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> >&)) // строки 111-113 - endl
114.          movl    $0, %eax                 // return EXIT_SUCCESS
115.          leave
116.          ret
117.    __static_initialization_and_destruction_0(int, int):
118.          pushq   %rbp
119.          movq    %rsp, %rbp
120.          subq    $16, %rsp
121.          movl    %edi, -4(%rbp)
122.          movl    %esi, -8(%rbp)
123.          cmpl    $1, -4(%rbp)
124.          jne     .L11
```

```
125.        cmpl    $65535, -8(%rbp)
126.        jne     .L11
127.        movl    $_ZStL8__ioinit, %edi
128.        call    std::ios_base::Init::Init() [complete object constructor]
129.        movl    $__dso_handle, %edx
130.        movl    $_ZStL8__ioinit, %esi
131.        movl    $_ZNSt8ios_base4InitD1Ev, %edi
132.        call    __cxa_atexit
133.    .L11:
134.        nop
135.        leave
136.        ret
137.    _GLOBAL__sub_I_CountLogarithm(double, long long):
138.        pushq   %rbp
139.        movq    %rsp, %rbp
140.        movl    $65535, %esi
141.        movl    $1, %edi
142.        call    __static_initialization_and_destruction_0(int, int)
143.        popq    %rbp
144.        ret
145.    LC1:
146.        .long   -400107883
147.        .long   1041313291
```

Приложение 1.3 Полный листинг программы на ассемблере x86-64 -O3

```
1.  CountLogarithm(double, long long):
2.          testq   %rdi, %rdi                  // Логическое и, результат никуда не
    записывается
3.          jle     .L13
4.          addq    $1, %rdi
5.          movl    $1, %eax
6.          pxor    %xmm1, %xmm1                // double result = 0
7.          jmp     .L12
8.  .L15:
9.          mulsd   %xmm0, %xmm0                // x *= x
10.         addq    $1, %rax
11.         addsd   %xmm2, %xmm1                // result +=
12.         cmpq    %rax, %rdi
13.         je      .L8
14. .L12:
15.         pxor    %xmm3, %xmm3                ////////////////////////////////////////////////////////////////
16.         movapd  %xmm0, %xmm2                //    Вычисление выражения                    //
17.         cvtsi2sdq       %rax, %xmm3         //    x / i                                   //
18.         divsd   %xmm3, %xmm2                ////////////////////////////////////////////////////////////////
19.         testb   $1, %al                     // Проверка остатка от деления на 2
20.         jne     .L15                        // переход если выполняется условие
21.         mulsd   %xmm0, %xmm0                // x *= x
22.         addq    $1, %rax
23.         subsd   %xmm2, %xmm1                // result -=
24.         cmpq    %rax, %rdi
25.         jne     .L12
26. .L8:
27.         movapd  %xmm1, %xmm0
28.         ret
29. .L13:
30.         pxor    %xmm1, %xmm1
31.         movapd  %xmm1, %xmm0
32.         ret
```

```
33. main:
34.         pushq   %rbx
35.         movq    %rsi, %rbx
36.         subq    $48, %rsp               // Сдвиг верхушки стека для локальных переменных
37.         movq    8(%rsi), %rdi           // Извлечь аргумент в %rdi
38.         xorl    %esi, %esi              // Обнуление регистра
39.         call    strtod
40.         movq    16(%rbx), %rdi          // Извлечь аргумент в %rdi
41.         xorl    %esi, %esi              // Обнуление регистра
42.         movl    $10, %edx
43.         movsd   %xmm0, 8(%rsp)
44.         call    strtoll
45.         leaq    16(%rsp), %rsi          // Параметр для call_gettime
46.         movl    $4, %edi                // Параметр для call_gettime
47.         movq    %rax, %rbx
48.         call    clock_gettime
49.         testq   %rbx, %rbx              // Сравнение n с нулем
50.         jle     .L21                    // Начало работы функции если n > 0
51.         movsd   8(%rsp), %xmm0
52.         leaq    1(%rbx), %rax
53.         pxor    %xmm1, %xmm1
54.         movl    $1, %edx
55.         jmp     .L20
56. .L24:
57.         mulsd   %xmm0, %xmm0            // x *= x
58.         addq    $1, %rdx                // Добавление единицы в цикле for
59.         addsd   %xmm2, %xmm1            // result +=
60.         cmpq    %rdx, %rax              // Сравнение i и n
61.         je      .L17
62. .L20:
63.         pxor    %xmm3, %xmm3            ////////////////////////////////////////////////////////////////////////
64.         movapd  %xmm0, %xmm2            //          Вычисление выражения                                  //
65.         cvtsi2sdq       %rdx, %xmm3     //          x / i                                                 //
66.         divsd   %xmm3, %xmm2            ////////////////////////////////////////////////////////////////////////
67.         testb   $1, %dl                 // Сравнение остатка от деления на 2 с единицей
```

```
68. jne    .L24
69.        mulsd   %xmm0, %xmm0        // x *= x
70.        addq    $1, %rdx            // Добавление единицы в цикле for
71.        subsd   %xmm2, %xmm1        // result -=
72.        cmpq    %rdx, %rax          // Сравнение i и n
73.        jne     .L20
74. .L17:
75.        movapd  %xmm1, %xmm0
76.        movl    $_ZSt4cout, %edi
77.           call     std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char,
    std::char_traits<char> >::_M_insert<double>(double)
78.        movq    %rax, %rdi
79.                 call        std::basic_ostream<char,  std::char_traits<char>  >&  std::endl<char,
    std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&) [clone .isra.0]
80.        leaq    32(%rsp), %rsi      ////////////////////////////////////////////////////////////////////////
81.        movl    $4, %edi            / clock_gettime(CLOCK_MONOTONIC_RAW,&end/
82.        call    clock_gettime       ////////////////////////////////////////////////////////////////////////
83.        movq    40(%rsp), %rax
84.        pxor    %xmm0, %xmm0        // Обнуление регистра для умножения
85.        subq    24(%rsp), %rax      // end.tv_nsec - start.tv_nsec
86.        cvtsi2sdq       %rax, %xmm0     // Преобразование из int в double
87.        pxor    %xmm1, %xmm1        // Обнуление регистра для сложения
88.        movq    32(%rsp), %rax
89.        subq    16(%rsp), %rax      // Первое вычитание
90.        mulsd   .LC1(%rip), %xmm0   // 1e-9 * (end.tv_nsec - start.tv_nsec)
91.        cvtsi2sdq       %rax, %xmm1     // Преобразование из int в double
92.        movl    $_ZSt4cout, %edi
93.        addsd   %xmm1, %xmm0        // Сложение для результата double time
94.           call     std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char,
    std::char_traits<char> >::_M_insert<double>(double)
95.        movq    %rax, %rdi
96. call                 std::basic_ostream<char,    std::char_traits<char>    >&    std::endl<char,
    std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&) [clone .isra.0]
97.           addq    $48, %rsp           // Смещение rsp для исключения локальных
    переменных
```

```
98.             xorl    %eax, %eax
99.         popq    %rbx
100.            ret
101.    .L21:
102.            pxor    %xmm1, %xmm1      // Обнуление регистра
103.            jmp     .L17
104.    _GLOBAL__sub_I_CountLogarithm(double, long long): ////////////////////////////////////////
105.            subq    $8, %rsp
106.            movl    $_ZStL8__ioinit, %edi
107.            call    std::ios_base::Init::Init() [complete object constructor]
108.            movl    $__dso_handle, %edx
109.            movl    $_ZStL8__ioinit, %esi
110.            movl    $_ZNSt8ios_base4InitD1Ev, %edi
111.            addq    $8, %rsp
112.            jmp     __cxa_atexit
113.    .LC1:
114.            .long   -400107883
115.            .long   1041313291
```