

**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**

**«название работы»**

**Студентки 2 курса, 24202 группы**

**Корсун Дарья Андреевны**

**Направление 09.03.01 – «Информатика и вычислительная техника»**

**Преподаватель:  
Кандидат технических наук  
В.А.Перепёлкин**

**Новосибирск 2025**



## **СОДЕРЖАНИЕ**

ЦЕЛЬ .....	4
ЗАДАНИЕ .....	4
ОПИСАНИЕ РАБОТЫ.....	5
ЗАКЛЮЧЕНИЕ .....	11
Приложение 1 (2, ...). Наименование приложения.....	12

## **ЦЕЛЬ**

1. Знакомство с программной архитектурой x86/x86-64.
2. Анализ ассемблерного листинга программы для архитектуры x86/x86-64.

## **ЗАДАНИЕ**

1. Изучить программную архитектуру x86/x86-64.
2. Для программы на языке Си (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры x86 и архитектуры x86-64, используя различные уровни комплексной оптимизации.
3. Проанализировать полученные листинги.
4. Составить отчет по лабораторной работе.

## 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:

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

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

Код программы

Проанализируем полученные на ассемблере код:

```
#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;
}
```

lab3\_o0\_x86-64.o: file format mach-o 64-bit x86-64

Disassembly of section \_\_TEXT,\_\_text:

```
0000000000000000 <_CalculateSin>:
 0: 55                      pushq %rbp           // Сохраняем указатель на
начало стек
 1: 48 89 e5                movq %rsp, %rbp   // Устанавливаем новый
базовый указатель
 4: f2 0f 11 45 f8          movsd %xmm0, -0x8(%rbp) // Сохраняем x в стек
 9: 48 89 7d f0              movq %rdi, -0x10(%rbp) // Сохраняем N в стек
 d: f2 0f 10 45 f8          movsd -0x8(%rbp), %xmm0 // x в xmm0
12: f2 0f 11 45 e8          movsd %xmm0, -0x18(%rbp) // term = x
17: f2 0f 10 45 f8          movsd -0x8(%rbp), %xmm0 // x в xmm0
1c: f2 0f 11 45 e0          movsd %xmm0, -0x20(%rbp) // result = x
21: 48 c7 45 d8 02 00 00 00  movq $0x2, -0x28(%rbp) // n = 2
29: 48 8b 45 d8              movq -0x28(%rbp), %rax // n в rax
2d: 48 3b 45 f0              cmpq -0x10(%rbp), %rax // Сравниваем n и N
31: 7f 6b                  jg 0x9e <_CalculateSin+0x9e> // n > N exit
33: f2 0f 10 45 f8          movsd -0x8(%rbp), %xmm0 // x
38: f2 0f 59 45 f8          mulsd -0x8(%rbp), %xmm0 // x*x
3d: 66 48 0f 7e c0          movq %xmm0, %rax // rax = x*x
42: 48 b9 00 00 00 00 00 80  movabsq $-0x8000000000000000, %rcx // маска для смены
знака
4c: 48 31 c8                xorq %rcx, %rax // x*x -> -(x*x)
4f: 66 48 0f 6e c0          movq %rax, %xmm0 // xmm0 = -(x*x)
54: 48 8b 45 d8              movq -0x28(%rbp), %rax // n в rax
58: 48 d1 e0                shlq %rax, %rax // n*2 (2*n)
5b: 48 83 e8 01              subq $0x1, %rax // 2*n - 1
5f: 48 8b 4d d8              movq -0x28(%rbp), %rcx // n в rax
63: 48 d1 e1                shlq %rcx, %rcx // n*2
```

```

66: 48 83 e9 02          subq $0x2, %rcx           // 2*n - 2
6a: 48 0f af c1          imulq%rcx, %rax        // (2*n-1)*(2*n-2)
6e: f2 48 0f 2a c8          cvtsi2sd %rax, %xmm1   // Конвертируем в double
73: f2 0f 5e c1          divsd%xmm1, %xmm0      // -(x*x)/((2*n-1)*(2*n-2))
77: f2 0f 59 45 e8          mulsd-0x18(%rbp), %xmm0 // term *= результат
7c: f2 0f 11 45 e8          movsd%xmm0, -0x18(%rbp) // Сохраняем term
81: f2 0f 10 45 e8          movsd-0x18(%rbp), %xmm0 // result += term
86: f2 0f 58 45 e0          addsd-0x20(%rbp), %xmm0 // result += term
8b: f2 0f 11 45 e0          movsd%xmm0, -0x20(%rbp) // Сохраняем result
90: 48 8b 45 d8          movq -0x28(%rbp), %rax // Загружаем n
94: 48 83 c0 01          addq $0x1, %rax        // n++
98: 48 89 45 d8          movq %rax, -0x28(%rbp) // Сохраняем n
9c: eb 8b                jmp 0x29 <_CalculateSin+0x29> // Возврат к началу цикла
9e: f2 0f 10 45 e0          movsd-0x20(%rbp), %xmm0 // Загружаем результат
a3: 5d                  popq %rbp           // Восстанавливаем указатель
a4: c3                  retq               // Возврат

```

После применения флага оптимизации можно заметить, что программа перестала обращаться к стеку, а стала использовать регистры для более быстрой работы, также цикл был развернут по четности/нечетности и соответственно кол-во итераций уменьшилось в два раза, так как за одну итерацию счетчик инкрементируется на 2, что уменьшает кол-во проверок

lab3\_o3\_x86-64.o: file format mach-o 64-bit x86-64

Disassembly of section \_\_TEXT,\_\_text:

```

0000000000000000 <_CalculateSin>:
    0: 48 83 ff 02          cmpq $0x2, %rdi           // ? N >= 2
    4: 0f 8c bd 00 00 00      jl 0xc7 <_CalculateSin+0xc7> // Если N < 2 -> return x
    a: 55                   pushq%rbp            // Сохраняем указ.
    b: 48 89 e5             movq %rsp, %rbp        // Устанавливаем указатель
    e: 66 0f 28 0d ba 00 00 00  movapd 0xba(%rip), %xmm1 // Маска для смены
знако
    16: 66 0f 57 c8          xorpd%xmm0, %xmm1      // -(x*x)
    1a: f2 0f 59 c8          mulsd%xmm0, %xmm1      // xmm1 = -(x*x)
    1e: 48 8d 4f ff          leaq -0x1(%rdi), %rcx     // rcx = N-1
    22: 48 83 ff 02          cmpq $0x2, %rdi           // Проверяем N=2
    26: 75 13                jne 0x3b <_CalculateSin+0x3b> // Если не N=2
    28: b8 04 00 00 00 00      movl $0x4, %eax         // Начинаем с n=4
(оптимизация)
    2d: 66 0f 28 d0          movapd %xmm0, %xmm2      // term = x
    31: f6 c1 01             testb$0x1, %cl        // Четность итераций
    34: 75 70                jne 0xa6 <_CalculateSin+0xa6> // Если нечетное,
обрабатываем последнюю итерацию
    36: e9 8b 00 00 00 00      jmp 0xc6 <_CalculateSin+0xc6> // Завершение
    3b: 48 89 ca             movq %rcx, %rdx        // rdx = количество итераций
    3e: 48 83 e2 fe          andq $-0x2, %rdx       // Округляем вниз до четного
числа
    42: b8 05 00 00 00 00      movl $0x5, %eax         // Начинаем с n=5
    47: 66 0f 28 d0          movapd %xmm0, %xmm2      // Сохраняем term = x
    4b: 0f 1f 44 00 00 00      nopl (%rax,%rax)
    50: 48 8d 70 fe          leaq -0x2(%rax), %rsi     // rsi = n-2
    54: 48 8d 78 fd          leaq -0x3(%rax), %rdi     // rdi = n-3
    58: 48 0f af f7          imulq%rdi, %rsi        // (n-2)*(n-3)
    5c: 0f 57 db              xorps%xmm3, %xmm3      // Обнуляем xmm3
    5f: f2 48 0f 2a de          cvtsi2sd %rsi, %xmm3   // Конвертируем в double
    64: 66 0f 28 e1          movapd %xmm1, %xmm4      // Копируем -(x*x)
    68: f2 0f 5e e3          divsd%xmm3, %xmm4      // -(x*x)/((n-2)*(n-3))
    6c: f2 0f 59 e2          mulsd%xmm2, %xmm4      // term *= результат пред

```

```

70: f2 0f 58 c4      addsd %xmm4, %xmm0           // result += term
74: 48 8d 70 ff      leaq -0x1(%rax), %rsi        // rsi = n-1
78: 48 0f af f0      imulq %rax, %rsi          // n*(n-1)
7c: 0f 57 db         xorps %xmm3, %xmm3          // xmm3 = 0
7f: f2 48 0f 2a de   cvtsi2sd %rsi, %xmm3       // Конвертируем в double
84: 66 0f 28 d1      movapd %xmm1, %xmm2        // Копируем -(x*x)
88: f2 0f 5e d3      divsd %xmm3, %xmm2        // -(x*x)/(n*(n-1))
8c: f2 0f 59 d4      mulsd %xmm4, %xmm2        // term *= результат
90: f2 0f 58 c2      addsd %xmm2, %xmm0        // result += term
94: 48 83 c0 04      addq $0x4, %rax          // n += 4
98: 48 83 c2 fe      addq $-0x2, %rdx         // Уменьшаем счетчик итераций
9c: 75 b2             jne 0x50 <_CalculateSin+0x50> // Продолжаем цикл
9e: 48 ff c8         decq %rax            // Корректируем n
a1: f6 c1 01         testb $0x1, %cl        // Проверяем нечетную
итерацию
a4: 74 20             je 0xc6 <_CalculateSin+0xc6> // Если четное, завершаем
a6: 48 8d 48 ff      leaq -0x1(%rax), %rcx      // Обработка последней
нечетной итерации
aa: 48 83 c0 fe      addq $-0x2, %rax
ae: 48 0f af c1      imulq %rcx, %rax
b2: 0f 57 db         xorps %xmm3, %xmm3
b5: f2 48 0f 2a d8   cvtsi2sd %rax, %xmm3
ba: f2 0f 5e cb      divsd %xmm3, %xmm1
be: f2 0f 59 d1      mulsd %xmm1, %xmm2
c2: f2 0f 58 c2      addsd %xmm2, %xmm0
c6: 5d               popq %rbp
c7: c3               retq

```

Теперь оценим листинги для 32 битной архитектуры.

lab3\_o0\_x86.o: file format mach-o 32-bit i386

Disassembly of section \_\_TEXT,\_\_text:

```

00000000 <_CalculateSin>:
 0: 55                 pushl %ebp           // Сохраняем базовый
указатель предыдущей функции
 1: 89 e5              movl %esp, %ebp // Устанавливаем новый
базовый указатель для текущей функции
 3: 83 ec 28           subl $0x28, %esp // Выделяем 40 байт в стеке
для локальных переменных
 6: e8 00 00 00 00     calll 0xb <_CalculateSin+0xb> // Вызов следующей инструкции
(POP) - трюк для получения PIC
  b: 58                popl %eax          // Получаем текущий адрес
(для Position Independent Code)
  c: 89 45 dc           movl %eax, -0x24(%ebp) // Сохраняем адрес для
доступа к данным
  f: 8b 45 10           movl 0x10(%ebp), %eax // Загружаем аргумент N
(лежит по адресу ebp+16)
 12: f2 0f 10 45 08    movsd 0x8(%ebp), %xmm0 // Загружаем аргумент x
(double по адресу ebp+8) в xmm0
 17: f2 0f 10 45 08    movsd 0x8(%ebp), %xmm0 // Повторно загружаем x
(избыточно)
 1c: f2 0f 11 45 f8    movsd %xmm0, -0x8(%ebp) // Сохраняем x в term
(локальная переменная)
 21: f2 0f 10 45 08    movsd 0x8(%ebp), %xmm0 // Снова загружаем x
 26: f2 0f 11 45 f0    movsd %xmm0, -0x10(%ebp) // Сохраняем x в result
(локальная переменная)
 2b: c7 45 ec 02 00 00 00 movl $0x2, -0x14(%ebp) // Инициализируем n = 2

```

```

32: 8b 45 ec          movl -0x14(%ebp), %eax      // Загружаем n в eax
35: 3b 45 10          cmpl 0x10(%ebp), %eax      // Сравниваем n с N
38: 7f 56             jg   0x90 <_CalculateSin+0x90> // Если n > N, выходим из
цикла
3a: 8b 45 dc          movl -0x24(%ebp), %eax      // Загружаем сохраненный
адрес для доступа к данным
3d: f2 0f 10 45 08    movsd 0x8(%ebp), %xmm0      // Загружаем x
42: f2 0f 59 c0        mulsd %xmm0, %xmm0       // Вычисляем x*x
46: 0f 28 88 a5 00 00 00 movaps 0xa5(%eax), %xmm1 // Загружаем маску для
смены знака [0x800000...]
4d: 66 0f ef c1        pxor %xmm1, %xmm0      // Меняем знак: xmm0 = -(x*x)
51: 8b 45 ec          movl -0x14(%ebp), %eax      // Загружаем n
54: d1 e0             shll %eax            // n*2 (2*n)
56: 83 e8 01          subl $0x1, %eax      // 2*n - 1
59: 8b 4d ec          movl -0x14(%ebp), %ecx      // Снова загружаем n
5c: d1 e1             shll %ecx            // n*2 (2*n)
5e: 83 e9 02          subl $0x2, %ecx      // 2*n - 2
61: 0f af c1          imull %ecx, %eax      // (2*n-1) * (2*n-2)
64: f2 0f 2a c8        cvtsi2sd %eax, %xmm1 // Конвертируем double
68: f2 0f 5e c1        divsd %xmm1, %xmm0      // -(x*x) / ((2*n-1)*(2*n-2))
6c: f2 0f 59 45 f8    mulsd -0x8(%ebp), %xmm0 // term *= результат
71: f2 0f 11 45 f8    movsd %xmm0, -0x8(%ebp) // Сохраняем term
76: f2 0f 10 45 f8    movsd -0x8(%ebp), %xmm0 // Загружаем term
7b: f2 0f 58 45 f0    addsd -0x10(%ebp), %xmm0 // result += term
80: f2 0f 11 45 f0    movsd %xmm0, -0x10(%ebp) // Сохраняем result
85: 8b 45 ec          movl -0x14(%ebp), %eax      // Загружаем n
88: 83 c0 01          addl $0x1, %eax      // n++
8b: 89 45 ec          movl %eax, -0x14(%ebp) // Сохраняем n
8e: eb a2             jmp  0x32 <_CalculateSin+0x32> // Возврат к началу цикла
90: f2 0f 10 45 f0    movsd -0x10(%ebp), %xmm0 // Загружаем result
95: f2 0f 11 45 e0    movsd %xmm0, -0x20(%ebp) // Сохраняем во временную
переменную
9a: dd 45 e0          fldl -0x20(%ebp)      // Загружаем результат в
регистр FPU
9d: 83 c4 28          addl $0x28, %esp      // Освобождаем стек
a0: 5d               popl %ebp           // Восстанавливаем базовый
указатель
a1: c3               retl                // Возврат из функции

```

Аналогично для уровня оптимизации -O3. Также как и в архитектуре x86-64 оптимизация достигнута за счет разворачивания цикла и работой с регистрами, вместо обращения к памяти.  
lab3\_o3\_x86.o: file format mach-o 32-bit i386

Disassembly of section \_\_TEXT,\_\_text:

```

00000000 <_CalculateSin>:
 0: 55                 pushl %ebp
 1: 89 e5              movl %esp, %ebp
 3: 57                 pushl %edi
 4: 56                 pushl %esi
 5: 83 ec 08            subl $0x8, %esp
 8: 8b 45 10            movl 0x10(%ebp), %eax
 b: f2 0f 10 45 08     movsd 0x8(%ebp), %xmm0
10: 83 f8 02            cmpl $0x2, %eax
13: 0f 8c ae 00 00 00  jl  0xc7 <_CalculateSin+0xc7>
19: e8 00 00 00 00     calll 0x1e <_CalculateSin+0x1e>

```

```

1e: 59          popl %ecx
1f: 66 0f 28 89 c2 00 00 00    movapd 0xc2(%ecx), %xmm1
27: 66 0f 57 c8          xorpd %xmm0, %xmm1
2b: f2 0f 59 c8          mulsd %xmm0, %xmm1
2f: 8d 48 ff           leal -0x1(%eax), %ecx
32: 83 f8 02           cmpl $0x2, %eax
35: 75 13             jne 0x4a <_CalculateSin+0x4a>
37: b8 04 00 00 00      movl $0x4, %eax
3c: 66 0f 28 d0           movapd %xmm0, %xmm2
40: f6 c1 01           testb $0x1, %cl
43: 75 66             jne 0xab <_CalculateSin+0xab>
45: e9 7d 00 00 00      jmp 0xc7 <_CalculateSin+0xc7>
4a: 89 ca             movl %ecx, %edx
4c: 83 e2 fe           andl $-0x2, %edx
4f: b8 05 00 00 00      movl $0x5, %eax
54: 66 0f 28 d0           movapd %xmm0, %xmm2
58: 0f 1f 84 00 00 00 00 00 00  nopl (%eax,%eax)
60: 8d 70 fe           leal -0x2(%eax), %esi
63: 8d 78 fd           leal -0x3(%eax), %edi
66: 0f af f7           imull%edi, %esi
69: 0f 57 db           xorps %xmm3, %xmm3
6c: f2 0f 2a de           cvtsi2sd %esi, %xmm3
70: 66 0f 28 e1           movapd %xmm1, %xmm4
74: f2 0f 5e e3           divsd %xmm3, %xmm4
78: f2 0f 59 e2           mulsd %xmm2, %xmm4
7c: f2 0f 58 c4           addsd %xmm4, %xmm0
80: 8d 70 ff           leal -0x1(%eax), %esi
83: 0f af f0           imull%eax, %esi
86: 0f 57 db           xorps %xmm3, %xmm3
89: f2 0f 2a de           cvtsi2sd %esi, %xmm3
8d: 66 0f 28 d1           movapd %xmm1, %xmm2
91: f2 0f 5e d3           divsd %xmm3, %xmm2
95: f2 0f 59 d4           mulsd %xmm4, %xmm2
99: f2 0f 58 c2           addsd %xmm2, %xmm0
9d: 83 c0 04           addl $0x4, %eax
a0: 83 c2 fe           addl $-0x2, %edx
a3: 75 bb             jne 0x60 <_CalculateSin+0x60>
a5: 48               decl %eax
a6: f6 c1 01           testb $0x1, %cl
a9: 74 1c             je 0xc7 <_CalculateSin+0xc7>
ab: 8d 48 ff           leal -0x1(%eax), %ecx
ae: 83 c0 fe           addl $-0x2, %eax
b1: 0f af c1           imull%ecx, %eax
b4: 0f 57 db           xorps %xmm3, %xmm3
b7: f2 0f 2a d8           cvtsi2sd %eax, %xmm3
bb: f2 0f 5e cb           divsd %xmm3, %xmm1
bf: f2 0f 59 d1           mulsd %xmm1, %xmm2
c3: f2 0f 58 c2           addsd %xmm2, %xmm0
c7: f2 0f 11 45 f0           movsd %xmm0, -0x10(%ebp)
cc: dd 45 f0           fldl -0x10(%ebp)
cf: 83 c4 08           addl $0x8, %esp
d2: 5e               popl %esi
d3: 5f               popl %edi
d4: 5d               popl %ebp
d5: c3               retl

```

Различия между предложенными архитектурами:

- 1) Кол-во регистров и их размерность ( 64 битные регистры начинающиеся на r) ( 32 битные регистры начинающиеся на e). Также в архитектуре x86-64 количество регистров общего назначения = 16, в то время как в x86 их 8)
- 2) Размер адресов и указателей ( в x86-64 выделяется 8 байт на указатель, в x86 4 байта соответственно)
- 3) Различия в работе с числами с плавающей точкой (возврат значений в x86-64 происходит через копирование регистра, а в x86 происходит предварительное копирование в FPU)

## **ЗАКЛЮЧЕНИЕ**

Был проанализирован ассемблерный листинг программы для архитектуры x86/x86-64.

Через ассемблерный листинг можно отследить изменения которые предлагают разные уровни компиляции и за счет которых как раз и происходит оптимизация кода. Как и предполагалось в предыдущей лабораторной улучшение производительности произошло за счет переноса переменных в регистры, что существенно уменьшило затраты времени на обращение к памяти. Также произошло развертывания цикла что также повысило эффективность за счет уменьшения количества лишних сравнений.

Если сравнивать предложенные архитектуры и листинги на них, можно сказать что они крайне похоже, разница состоит в отличающихся названиях, размере адресов, небольшими отличиями в работе с числами с плавающей точкой и количестве регистров (хоть в данной лабораторной это и не сказалось на результатах, в связи с маленьким количеством используемых переменных)

## **Приложение 1 (2, ...). Наименование приложения**

*В случае, если в ходе выполнения лабораторной работы необходимо было написать одну или несколько компьютерных программ, скриптов и т.п., то в приложениях приводятся листинги данных объектов.*