###### 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 x86/x86-64»

Студентки \_\_\_2\_\_\_\_ курса, \_22208\_ группы

**Жуйко Снежаны Евгеньевны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю.Кудинов

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](file:///D:\1NSU\ECM\lab_template.doc#_Toc18443921)

[ЗАДАНИЕ 3](file:///D:\1NSU\ECM\lab_template.doc#_Toc18443922)

[ОПИСАНИЕ РАБОТЫ 5](file:///D:\1NSU\ECM\lab_template.doc#_Toc18443923)

[ЗАКЛЮЧЕНИЕ 9](file:///D:\1NSU\ECM\lab_template.doc#_Toc18443924)

# 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:*

* 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 лабораторной работы 1) сгенерировать*

*ассемблерные листинги для архитектуры x86 и архитектуры x86-64,*

*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*

*архитектур x86 и x86-64 на конкретных участках ассемблерного*

*кода.*

* 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 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

# *Листинг программы:*

#include <stdio.h>

double expp(long long int countN, double x)

{

  double result = 1;

  double t = 1;

  for (long long int n = 1; n < countN; ++n)

  {

    t \*= x/n;

    result += t;

  }

  return result;

}

int main()

{

  long long int countN = 2500000000;

  double x = 3;

  printf("my result: %f\n", expp(countN, x));

}

Команда компиляции: gcc expp.c –o expp -O0

# *Листинг ассемблера:*

gcc expp.c -S –o expp.s -O0

expp(long long, double):

        pushq   %rbp //загружает адрес rbp на вершину стека, чтобы запомнить старое значение стека

        movq    %rsp, %rbp //записываем в rbp адрес rsp, чтобы определить начало нового кадра, здесь от rsp не вычитается никакое число, т.е. память под локальные переменные в кадре не выделяется, так как никакая другая функция в expp не вызывается. Стек резервируется для того, чтобы потом восстановить кадр предыдущей подпрограммы, так как потом там могут быть записаны другие локальные переменные, которые затрутся после выхода из функции

        movq    %rdi, -40(%rbp) //записываем значение из %rdi в -40(%rbp)(Из адреса в rbp вычитаем 40. Стек растет вниз, поэтому локальные переменные располагаются по адресу %rbp – х, а параметры по адресу %rbp – х.

        movsd   %xmm0, -48(%rbp)

        movsd   .LC0(%rip), %xmm0

        movsd   %xmm0, -8(%rbp)

        movsd   .LC0(%rip), %xmm0

        movsd   %xmm0, -16(%rbp)

        movq    $1, -24(%rbp)

.L3:

        movq    -24(%rbp), %rax

        cmpq    -40(%rbp), %rax

        jge     .L2

        cvtsi2sdq       -24(%rbp), %xmm1

        movsd   -48(%rbp), %xmm0

        divsd   %xmm1, %xmm0

        movsd   -16(%rbp), %xmm1

        mulsd   %xmm1, %xmm0

        movsd   %xmm0, -16(%rbp)

        movsd   -8(%rbp), %xmm0

        addsd   -16(%rbp), %xmm0

        movsd   %xmm0, -8(%rbp)

        addq    $1, -24(%rbp)

        jmp     .L3

.L2:

        movsd   -8(%rbp), %xmm0 //eax хранит целочисленный результат функции, а xmm0 - вещественный

        popq    %rbp

        ret //возврат из подпрогрммы

.LC2:

        .string "my result: %f\n"

main:

        pushq   %rbp

        movq    %rsp, %rbp

        subq    $16, %rsp //резервирует память под локальные переменные

        movl    $2500000000, %eax

        movq    %rax, -8(%rbp)

        movsd   .LC1(%rip), %xmm0 //%rip – регистр instructor pointer

        movsd   %xmm0, -16(%rbp)

        movq    -16(%rbp), %rdx

        movq    -8(%rbp), %rax

        movq    %rdx, %xmm0

        movq    %rax, %rdi

        call    expp(long long, double)

        movl    $.LC2, %edi

        movl    $1, %eax

        call    printf

        movl    $0, %eax

        leave

        ret

.LC0:

        .long   0

        .long   1072693248

.LC1:

        .long   0

        .long   1074266112

gcc expp.c -m32 -S –o expp.s -O0

expp(long long, double):

        pushl   %ebp

        movl    %esp, %ebp

        subl    $48, %esp

        movl    8(%ebp), %eax

        movl    %eax, -40(%ebp)

        movl    12(%ebp), %eax

        movl    %eax, -36(%ebp)

        movl    16(%ebp), %eax

        movl    %eax, -48(%ebp)

        movl    20(%ebp), %eax

        movl    %eax, -44(%ebp)

        fld1

        fstpl   -8(%ebp)

        fld1

        fstpl   -16(%ebp)

        movl    $1, -24(%ebp)

        movl    $0, -20(%ebp)

.L3:

        movl    -24(%ebp), %eax

        movl    -20(%ebp), %edx

        cmpl    -40(%ebp), %eax

        movl    %edx, %eax

        sbbl    -36(%ebp), %eax

        jge     .L2

        fildq   -24(%ebp) //загрузить целое число

        fldl    -48(%ebp)

        fdivp   %st, %st(1)

        fldl    -16(%ebp)

        fmulp   %st, %st(1)

        fstpl   -16(%ebp)

        fldl    -8(%ebp)

        faddl   -16(%ebp)

        fstpl   -8(%ebp)

        addl    $1, -24(%ebp)

        adcl    $0, -20(%ebp) для работы с числами которые не помещаются в один регистр, добавляют флаг переноса

        jmp     .L3

.L2:

        fldl    -8(%ebp)

        leave

        ret

.LC3:

        .string "my result: %d\n"

main:

        leal    4(%esp), %ecx

        andl    $-16, %esp //выравнивание стека

        pushl   -4(%ecx)

        pushl   %ebp

        movl    %esp, %ebp

        pushl   %ecx

        subl    $20, %esp

        movl    $250000000, -16(%ebp)

        movl    $0, -12(%ebp)

        fldl    .LC2

        fstpl   -24(%ebp)

        pushl   -20(%ebp)

        pushl   -24(%ebp)

        pushl   -12(%ebp)

        pushl   -16(%ebp)

        call    expp(long long, double)

        addl    $16, %esp

        subl    $4, %esp

        leal    -8(%esp), %esp

        fstpl   (%esp)

        pushl   $.LC3

        call    printf

        addl    $16, %esp

        movl    $0, %eax

        movl    -4(%ebp), %ecx

        leave

        leal    -4(%ecx), %esp

        ret

.LC2:

        .long   0

        .long   1074266112

# ЗАКЛЮЧЕНИЕ

*С ключом оптимизации -01 компилятор проверяет условие цикла с самого начала программы до for, с ключом оптимизации -02 компилятор перемешивает команды, встраивает функции. В 32-разрядной архитектуре используется стек FPU, числа, которые не помещаются в один регистр общего назначения, делятся на две части – младшую и старшую. В зависимости от уровня оптимизации операции с младшей и старшей частью происходят или только на стеке FPU, или с помощью стека FPU и регистров общего назначения. При этом происходит проверка флага переполнения при арифметических операциях.*