###### 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»

студента (ки) 2 курса, 18205 группы

Гайдамака Андрея Владиславовича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

(ученая степень, звание)

А. Ю. Власенко

Новосибирск 2019

ЦЕЛЬ 3

ЗАДАНИЕ 3

ОПИСАНИЕ РАБОТЫ 3

ЗАКЛЮЧЕНИЕ 4

Приложение 1. Код программы epower 5

Приложение 2. Ассемблерный листинг……………………………………7

# 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.

**Вариант задания: №5**

# ОПИСАНИЕ РАБОТЫ

После изучения программной архитектуры х86/х86-64, был проанализирован ассемблерный листинг (Приложение 2) программы на языке Си (из лабораторной работы №1)(Приложение 1). Во время разбора были сопоставлены инструкции на языке Си с машинными программами. Так же было определено размещение переменных, регистры в которых передаются аргументы в функцию и возвращаются из нее. На данном ассемблерном коде была изучена работа с вещественными числами, со стеком.

# ЗАКЛЮЧЕНИЕ

Можно сделать вывод, что рассмотрение ассемблерного листинга(Приложение 2) программы помогает в разборе работы функций, циклов, условий на очень низком уровне, что сильно помогает в оптимизации программы. Так же очень важным преимуществом изучения ассемблера является возможность вставки, ассемблерного кода в критически важные по производительности участки кода, при этом важно чтобы участок ассемблерного кода и код всей программы следовал единому ABI (Application Binary Interface)

# Приложение 1. Код программы epower

*#include <stdio.h>*

*#include <stdlib.h>*

*int factorial(const int n){*

*int res = 1;*

*for (int i = 1; i <= n; ++i){*

*res \*= i;*

*}*

*return res;*

*}*

*int power(int a, int n){*

*int res = 1;*

*for (int i = 0; i < n; ++i){*

*res \*= a;*

*}*

*return res;*

*}*

*double epower(const long long int n, const long long int x){*

*double epow = 0;*

*double last = 0;*

*double tmp = 0;*

*for(long int i = 0; i <= n; ++i){*

*if (i <= 7){*

*last = (double)power(x, i) / factorial(i);*

*epow += last;*

*}*

*else{*

*last \*= (double)((double)x / i);*

*epow += last;*

*}*

*}*

*return epow;*

*}*

*int main(int argc, char \*\* argv) {*

*if (argc != 3){*

*return 1;*

*}*

*long long int n = atoll(argv[1]);*

*long long int x = atoll(argv[2]);*

*double epow = epower(n, x);*

*printf("%f\n", epow);*

*return 0;*

*}*

**Приложение 2. Ассемблерный Листинг**

*\_Z9factoriali:*

*pushq %rbp*

*movq %rsp, %rbp*

*movl %edi, -20(%rbp) //n – передается в edi и хранится в -20(%rbp)*

*movl $1, -4(%rbp) // res хранится в -4(%rbp)*

*movl $1, -8(%rbp) //i хранится в -8(%rbp)*

*.L3: //начало цикла*

*movl -8(%rbp), %eax*

*cmpl -20(%rbp), %eax*

*jg .L2 //jg – строго больше*

*movl -4(%rbp), %eax*

*imull -8(%rbp), %eax*

*movl %eax, -4(%rbp)*

*addl $1, -8(%rbp)*

*jmp .L3*

*.L2:*

*movl -4(%rbp), %eax*

*popq %rbp*

*ret*

*\_Z5powerii:*

*pushq %rbp*

*movq %rsp, %rbp*

*movl %edi, -20(%rbp) )//a – передается в edi и хранится в -20(%rbp)*

*movl %esi, -24(%rbp) ) //n – передается в esi и хранится в -20(%rbp)*

*movl $1, -4(%rbp) // res хранится в -4(%rbp)*

*movl $0, -8(%rbp) // i хранится в -8(%rbp)*

*.L7:*

*movl -8(%rbp), %eax*

*cmpl -24(%rbp), %eax*

*jge .L6 //jge – больше и равно*

*movl -4(%rbp), %eax*

*imull -20(%rbp), %eax*

*movl %eax, -4(%rbp)*

*addl $1, -8(%rbp)*

*jmp .L7*

*.L6:*

*movl -4(%rbp), %eax*

*popq %rbp*

*ret*

*\_Z6epowerxx:*

*pushq %rbp*

*movq %rsp, %rbp*

*subq $56, %rsp*

*movq %rdi, -40(%rbp) //n – передается в rdi и хранится в -20(%rbp)*

*movq %rsi, -48(%rbp) //x – передается в rsi и хранится в -20(%rbp)*

*pxor %xmm0, %xmm0 // получение нуля в xmm0*

*movsd %xmm0, -8(%rbp) // epow хранится в -8(%rbp)*

*pxor %xmm0, %xmm0*

*movsd %xmm0, -16(%rbp) // last хранится в -16(%rbp)*

*pxor %xmm0, %xmm0*

*movsd %xmm0, -32(%rbp) // tmp хранится в -32(%rbp)*

*movq $0, -24(%rbp) // // i хранится в -24(%rbp)*

*.L13:*

*movq -24(%rbp), %rax*

*cmpq -40(%rbp), %rax*

*jg .L10*

*cmpq $7, -24(%rbp)*

*jg .L11*

*movq -24(%rbp), %rax*

*movl %eax, %edx*

*movq -48(%rbp), %rax*

*movl %edx, %esi*

*movl %eax, %edi*

*call \_Z5powerii*

*pxor %xmm2, %xmm2*

*cvtsi2sd %eax, %xmm2 //convert scalar int to scalar double*

*movsd %xmm2, -56(%rbp)*

*movq -24(%rbp), %rax*

*movl %eax, %edi*

*call \_Z9factoriali*

*pxor %xmm0, %xmm0*

*cvtsi2sd %eax, %xmm0*

*movsd -56(%rbp), %xmm2*

*divsd %xmm0, %xmm2*

*movapd %xmm2, %xmm0*

*movsd %xmm0, -16(%rbp)*

*movsd -8(%rbp), %xmm0*

*addsd -16(%rbp), %xmm0*

*movsd %xmm0, -8(%rbp)*

*jmp .L12*

*.L11:*

*pxor %xmm0, %xmm0*

*cvtsi2sdq -48(%rbp), %xmm0*

*pxor %xmm1, %xmm1*

*cvtsi2sdq -24(%rbp), %xmm1*

*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)*

*.L12:*

*addq $1, -24(%rbp)*

*jmp .L13*

*.L10:*

*movsd -8(%rbp), %xmm0*

*leave*

*ret*

*main:*

*pushq %rbp*

*movq %rsp, %rbp*

*subq $48, %rsp*

*movl %edi, -36(%rbp) //argc – передается в edi и хранится в -20(%rbp)*

*movq %rsi, -48(%rbp) //argv – передается в rsi и хранится в -20(%rbp)*

*cmpl $3, -36(%rbp)*

*je .L16*

*movl $1, %eax*

*jmp .L17*

*.L16:*

*movq -48(%rbp), %rax*

*addq $8, %rax*

*movq (%rax), %rax*

*movq %rax, %rdi*

*call atoll*

*movq %rax, -8(%rbp)*

*movq -48(%rbp), %rax*

*addq $16, %rax*

*movq (%rax), %rax*

*movq %rax, %rdi*

*call atoll*

*movq %rax, -16(%rbp)*

*movq -16(%rbp), %rdx*

*movq -8(%rbp), %rax*

*movq %rdx, %rsi*

*movq %rax, %rdi*

*call \_Z6epowerxx*

*movq %xmm0, %rax*

*movq %rax, -24(%rbp)*

*movl $0, %eax*

*.L17:*

*leave*

*ret*