###### 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 2 курса, 21202 группы

**Куращенко Льва Владиславовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Перепелкин В.А.

Новосибирск 2022

1. **Цель**

1. Изучить аспекты работы с языком ассемблера;

2. Ознакомиться с архитектурой x86/x86-64;

3. Провести детальный анализ ассемблерного кода программы 1й лабораторной работы.

1. **Задание**

Изучить программную архитектуру x86/x86-64. Ознакомиться с набором регистров процессора, основными арифметико-логическими командами, командами арифметического сопроцессора. Изучить способы адресации памяти, работу со стеком, так же понять принцип вызова подпрограмм, передачу параметров в подпрограммы и возврат результатов. Для программы на языке С++ сгенерировать ассемблерные листинги для архитектуры x86 и архитектуры x86-64, после чего провести анализ ассемблерных листингов, сопоставляя команды языка С++ с машинными командами. После детального анализа процессорных инструкций продемонстрировать использование ключевых особенностей архитектур x86 и x86-64 на конкретных участках кода, после чего сравнить различия в программах для архитектуры x86 и архитектуры x86-64.

1. **Листинг программы**

#include <stdio.h>  
int main(){  
 long long int N = 2100000000;  
 double pi = 0;  
 double delta = 0;  
 for (int i = 0; i < N; i++){  
 delta = (double)4/(double)(2\*i+1);  
 if (i % 2 == 1)  
 delta = -delta;  
 pi += delta;  
 }  
 printf("%lf",pi);  
 return 0;  
}

1. **Ассемблерный листинг для архитектуры x86-64**

.LC3:  
 .string "%lf" //метка на форматную строку для вывода  
main:  
 pushq %rbp //Выделяем стековый кадр  
 movq %rsp, %rbp //копируем адрес вершины стека в rbp  
 subq $32, %rsp //сдвигаем вершину стека на 32 байта под наши переменные  
 movq $2100000000, -32(%rbp) //Записываем на место прошлой вершины стека 8 байт значение N = 2100000000  
 pxor %xmm0, %xmm0 // Обнуляем xmm0  
 movsd %xmm0, -8(%rbp) //Заполняем 8 байт нулями для double pi = 0  
 pxor %xmm0, %xmm0 // Обнуляем xmm0  
 movsd %xmm0, -16(%rbp) //Заполняем 8 байт нулями для double delta= 0  
 movl $0, -20(%rbp) // заполняем 4 байта нулями для int i = 0  
 jmp .L2 //Переходим на метку проверку условий цикла  
  
//Тело цикла  
.L4:  
 movl -20(%rbp), %eax //Загружаем значение i в %eax  
 addl %eax, %eax //Умножаем eax на два  
 addl $1, %eax //Добавляем единичку в %eax  
 pxor %xmm1, %xmm1 //Зануляем %xmm1  
 cvtsi2sdl %eax, %xmm1 //Преобразуем в double и помещаем в %xmm1  
 movsd .LC1(%rip), %xmm0 //Кидаем в %xmm0 значение 0 из .LC1   
 divsd %xmm1, %xmm0 //Делим %xmm0 на %xmm1  
 movsd %xmm0, -16(%rbp) //Копируем полученное значение в delta  
 movl -20(%rbp), %edx //Копируем i в %edx  
 movl %edx, %eax //Копируем i в %eax  
 sarl $31, %eax  
  
 //Проверка на четность  
 shrl $31, %eax   
 addl %eax, %edx  
 andl $1, %edx  
 subl %eax, %edx  
 movl %edx, %eax  
 //Если четно, идем на прибавление к pi  
 cmpl $1, %eax  
 jne .L3  
  
 //Если нечетно, отнимаем дельту  
 movsd -16(%rbp), %xmm0 //Копируем в xmm0 delta  
 movq .LC2(%rip), %xmm1  
  
 //Xor'им с 10...00, получаем -delta, присваиваем delta  
 xorpd %xmm1, %xmm0  
 movsd %xmm0, -16(%rbp)  
  
//Прибавляем к pi delta  
.L3:  
 movsd -8(%rbp), %xmm0  
 addsd -16(%rbp), %xmm0  
 movsd %xmm0, -8(%rbp)  
 addl $1, -20(%rbp) //Инкремент счетчика  
//Проверка условий цикла  
.L2:  
 movl -20(%rbp), %eax //Кладем в eax текущее значение i  
 cltq //Приводим 4 байтовый i к 8 байтам в rax  
 cmpq %rax, -32(%rbp) //Сравниваем N и i  
 jg .L4 //Если i меньше N, заходим в тело цикла .L4  
 movq -8(%rbp), %rax //Копируем pi в rax  
 movq %rax, %xmm0 //Копируем pi в xmm0, т.к. выводим вещественное число  
 movl $.LC3, %edi //Копируем форматную строку в edi, потому что первый аргумент в printf  
 movl $1, %eax // пишем 1 в eax, т.к. выводим только pi  
 call printf //Выводим pi  
 movl $0, %eax //Зануляем eax  
 leave //Выходим из main  
 ret //Возврат из main  
 //Завершение программы  
  
//Константы  
.LC1:  
 .long 0  
 .long 1074790400  
.LC2:  
 .long 0  
 .long -2147483648  
 .long 0  
 .long 0

1. **Ассемблерный листинг для архитектуры x86**

.LC2:  
 .string "%lf"  
main:  
 leal 4(%esp), %ecx //В ecx копируем последние 4 байта перед вершиной стека  
 andl $-16, %esp  
 pushl -4(%ecx)  
 pushl %ebp  
 movl %esp, %ebp  
 pushl %ecx  
 subl $52, %esp  
 movl $2100000000, -40(%ebp) //Записали N  
 movl $0, -36(%ebp)  
 fldz //Загрузка нуля в вершину стека  
 fstpl -16(%ebp) //Запись вершины стека с выталкиванием  
 fldz  
 fstpl -24(%ebp)  
 movl $0, -28(%ebp)  
 jmp .L2  
.L4:  
 movl -28(%ebp), %eax  
 addl %eax, %eax  
 addl $1, %eax  
 movl %eax, -44(%ebp)  
 fildl -44(%ebp) //Загрузка в веришну стека  
 fldl .LC1 //Подгрузили константу  
 fdivp %st, %st(1) //Поделили, вытолкнули из стека  
 fstpl -24(%ebp) //Записали, вытолкнули  
 movl -28(%ebp), %edx  
 movl %edx, %eax  
 sarl $31, %eax //Сдвиги  
 shrl $31, %eax  
 addl %eax, %edx  
 andl $1, %edx  
 subl %eax, %edx  
 movl %edx, %eax  
 cmpl $1, %eax  
 jne .L3 //Условие на запись -detla  
 fldl -24(%ebp)  
 fchs //Изменение знака  
 fstpl -24(%ebp)  
.L3:  
 fldl -16(%ebp)  
 faddl -24(%ebp)  
 fstpl -16(%ebp)  
 addl $1, -28(%ebp)  
.L2:  
 movl -28(%ebp), %eax  
 cltd  
 cmpl -40(%ebp), %eax  
 movl %edx, %eax  
 sbbl -36(%ebp), %eax  
 jl .L4  
 subl $4, %esp  
 pushl -12(%ebp)  
 pushl -16(%ebp)  
 pushl $.LC2  
 call printf  
 addl $16, %esp  
 movl $0, %eax  
 movl -4(%ebp), %ecx  
 leave  
 leal -4(%ecx), %esp  
 ret  
.LC1:  
 .long 0  
 .long 1074790400

1. **Ассемблерный листинг для архитектуры x86-64 с оптимизацией Ofast**

.LC7:  
 .string "%lf"  
main:  
//В этом блоке идет применение sse2 инструкций и запись констант из .LC#  
 subq $8, %rsp //Сдвигаем вершину стэка на 8 байт  
 movsd .LC4(%rip), %xmm6 //0, 1074790400  
 xorl %eax, %eax  
 movsd .LC6(%rip), %xmm5 //0, 10..0  
 movdqa .LC0(%rip), %xmm2 //0, 1, 2, 3  
 movdqa .LC2(%rip), %xmm7 //1,1,1,1  
 pxor %xmm3, %xmm3 //Зануляем xmm3, xmm4  
 pxor %xmm4, %xmm4  
 movdqa .LC1(%rip), %xmm8 //4,4,4,4  
 unpcklpd %xmm6, %xmm6 //Инструкция sse2 на перенос нижних упакованных значений double с чередованием  
 unpcklpd %xmm5, %xmm5  
  
//Этот блок повторяет то, что выполняется в теле главного цикла в O0, но в развернутом цикле на 2 векторах по два значения и со счетчиком N/4  
.L2:  
 //Копируем значения  
 movdqa %xmm2, %xmm0  
 movapd %xmm6, %xmm10  
 movdqa %xmm4, %xmm11  
  
 //Прибавляем счетчик  
 addl $1, %eax  
  
 //Копируем значения  
 movdqa %xmm0, %xmm1  
  
 //Побитовое и, побитовое сложение,  
 //сдвиг влево, сравнение равенства,  
 //преобразование dword в double, деление,  
 //перераспределние упакованных слов и по новой  
 pand %xmm7, %xmm0 //Упакованное обитовое сложение  
 paddd %xmm8, %xmm2 //Упакованное сложение  
 pslld $1, %xmm1 //Cдвиг влево упакованных двойных словь  
 pcmpeqd %xmm4, %xmm0 //Сравнение на равенство упакованных слов  
 paddd %xmm7, %xmm1 //Прибавляем  
 cvtdq2pd %xmm1, %xmm9 //Преобразование двух dword в double  
 divpd %xmm9, %xmm10 //векторное деление double  
 pshufd $238, %xmm1, %xmm1  
 movapd %xmm6, %xmm9 //перемещение упакованных double  
 cvtdq2pd %xmm1, %xmm1  
 pcmpgtd %xmm0, %xmm11 //Сравнение на больше  
 divpd %xmm1, %xmm9 //Векторное деление  
 movdqa %xmm0, %xmm1  
 punpckhdq %xmm11, %xmm0 //Распаковка в qword  
 punpckldq %xmm11, %xmm1  
 movapd %xmm10, %xmm12 //перемещение слов  
 andpd %xmm1, %xmm10 //упакованное и  
 xorpd %xmm5, %xmm12 //упакованный xor  
 andnpd %xmm12, %xmm1 //упакованный и  
 orpd %xmm10, %xmm1 //упакованный |  
 movapd %xmm9, %xmm10  
 andpd %xmm0, %xmm9  
 xorpd %xmm5, %xmm10  
 andnpd %xmm10, %xmm0  
 orpd %xmm9, %xmm0  
 addpd %xmm0, %xmm1  
 addpd %xmm1, %xmm3  
  
 //Перемещение по метке в цикл  
 cmpl $525000000, %eax  
 jne .L2  
  
 //Вывод числа  
 movapd %xmm3, %xmm0  
 movl $.LC7, %edi  
 movl $1, %eax  
 unpckhpd %xmm3, %xmm0  
 addpd %xmm3, %xmm0  
 call printf  
 xorl %eax, %eax  
 addq $8, %rsp  
 ret  
.LC0:  
 .long 0  
 .long 1  
 .long 2  
 .long 3  
.LC1:  
 .long 4  
 .long 4  
 .long 4  
 .long 4  
.LC2:  
 .long 1  
 .long 1  
 .long 1  
 .long 1  
.LC4:  
 .long 0  
 .long 1074790400  
.LC6:  
 .long 0  
 .long -2147483648