###### 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, группы 19201

Хаецкой Дарьи Владимировны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2020

1. **Цель**

Изучить аспекты работы с языком ассемблера, ознакомиться с архитектурой x86-64 и провести детальный анализ ассемблерного кода программы первой лабораторной работы.

1. **Задание**

Изучить программную архитектуру 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 x86-64, после чего провести анализ ассемблерных листингов, сопоставляя команды языка С с машинными командами. После анализа процессорных инструкций продемонстрировать использование ключевых особенностей архитектуры x86-64 на конкретных участках кода, после чего сравнить различия в программах с оптимизациями и без.

**Листинг программы**

**#include <stdlib.h>**

**#include <math.h>**

**#include <time.h>**

**#define pi 3.14159265**

**#define N 200000000**

**long double sinus(double x){**

**long double current, prev = x, sin = 0;**

**long double numenator = 0, denumenator = 0;**

**for (int n = 1, f = 1; n < N; n++, f+=2){**

**sin += prev;**

**prev = abs(prev);**

**numenator = pow(-1, n) \* pow(x,2);**

**denumenator = (f + 1) \* (f + 2);**

**current = prev \* numenator / denumenator;**

**prev = current;**

**}**

**return sin;**

**}**

**int main() {**

**struct timespec start, end;**

**int degrees = 90;**

**clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);**

**sinus(toRadians(degrees));**

**clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);**

**double time = end.tv\_sec-start.tv\_sec + 1e-9 \* (end.tv\_nsec - start.tv\_nsec);**

**printf("%ld \n", time);**

**return 0;**

**}**

**Ассемблерный листинг для архитектуры x86-64**

toRadians:

pushq %rbp // %rbp — это указатель базы, который указывает на базу текущего стекового фрейма

movq %rsp, %rbp // %rsp — указатель стека, который указывает на вершину текущего стекового фрейма.

//записываем старый указатель базы в стек, чтобы сохранить его на будущее. Потом

//копируем значение указателя стека в указатель базы.

movsd %xmm0, -8(%rbp) // Эта инструкция копирует degrees в %rbp

movsd -8(%rbp), %xmm1 // Записать в xmm1 значение из памяти. по сути создали локальную переменную

// и сохранили туда аргумент функции xmm1 - регистр математического

// сопроцессора для хранения чисел с плавающей

// точкой. находится в SSE (Streaming SIMD Extensions)

movsd .LC0(%rip), %xmm0 // из LC0 выгружается значение pi/180 в освободившийся регистр

mulsd %xmm1, %xmm0 // pi/180 \* degrees

movq %xmm0, %rax // помещаем подсчитанное в регистр возвр. значения

movq %rax, %xmm0

popq %rbp // Мы достаем старый указатель базы и помещаем его обратно

//в %rbp, а затем инструкция ret перебрасывает нас к адресу возвращения,

ret //который тоже хранится в стековом фрейме

sinus:

pushq %rbp

movq %rsp, %rbp

subq $96, %rsp

movsd %xmm0, -72(%rbp)

fldl -72(%rbp) // Помещаем в стек сопроцессора, т.е. st(0), переменную

fstpt -16(%rbp) // с помощью fstp происходит обмен данными между ячейками памяти и регистрами сопроцессора

fldz // Push +0.0 onto the FPU register stack

fstpt -32(%rbp) // store 80-bit real and pop stack

movl $1, -36(%rbp) // в f и n записываются единицы

movl $1, -40(%rbp)

jmp .L4 // Безусловный переход на метку .L4

.L5:

fldt -32(%rbp) // загружаем sin

fldt -16(%rbp) // и prev

faddp %st, %st(1) // прибавляем к sin prev и сохраняем значение в st(0)

fstpt -32(%rbp) // Записываем в sin значение из st(0)

fldt -16(%rbp) // округление

fabs

fstpt -128(%rbp)

movq -128(%rbp), %rax

movl -120(%rbp), %edx

movq %rax, -16(%rbp)

movl %edx, -8(%rbp)

pxor %xmm0, %xmm0 // освободить регистры

cvtsi2sdl -36(%rbp), %xmm0 // приведение int к double числа n

movq .LC2(%rip), %rax // в rax записывается -1

movapd %xmm0, %xmm1 // n помещается в xmm1 (Перемещение упакованных выровненных Double)

movq %rax, %xmm0 // -1 помещается в xmm0

call pow // вызов возведения в степень, аргументы берутся из регистров

movsd %xmm0, -120(%rbp) // возвращенное значение записывается в ячейку -120(%rbp)

movsd .LC3(%rip), %xmm0 // записывается двойка

movq -104(%rbp), %rax // и х

movapd %xmm0, %xmm1 // помещаются в нужные регистры

movq %rax, %xmm0

call pow // после чего вызывается возведение

movapd %xmm0, %xmm2 // результат вычисления записываем в xmm2

mulsd -120(%rbp), %xmm2 // перемножаем результаты

movsd %xmm2, -120(%rbp) // запоминаем произведение в ячейку -120(%rbp)

fldl -120(%rbp) // Помещаем в стек сопроцессора

fstpt -64(%rbp) // store 80-bit real and pop stack. записали результат.

movl -40(%rbp), %eax // копируем f в eax

leal 1(%rax), %edx // прибавляем единицу к значению в rax и записываем в edx

movl -40(%rbp), %eax // снова копируем f в eax

addl $2, %eax // прибавляем 2 к значению в eax (f)

imull %edx, %eax // перемножаем содержимое edx и eax

movl %eax, -120(%rbp) // пишем вычисленное значение в 120 ячейку

fildl -120(%rbp) // кладем на стек

fstpt -80(%rbp) // выталкиваем и записываем в 80-ую

fldt -16(%rbp) // загружаем prev на стек

fldt -64(%rbp) // загружаем numenamor

fmulp %st, %st(1) // перемножаем их

fldt -80(%rbp) // загружаем denominator

fdivrp %st, %st(1) // делим произведение на него

fstpt -96(%rbp) // пишем вычисленное в current

fldt -96(%rbp) // в этих двух строчках записываем current в prev

fstpt -16(%rbp)

addl $1, -36(%rbp) // инкремент f и n

addl $2, -40(%rbp)

.L4: // продолжение sinus

cmpl $199999999, -36(%rbp) // n сравнивается с константой N

jle .L5 // jump if the previous comparison is "less"

fldt -32(%rbp) // Load Floating Point Value. T.е вернётся синус

leave // Восстанавливаем указатели стека и кадра

ret // эти две команды передают управление вызвавшей процедуре

main:

pushq %rbp

movq %rsp, %rbp

subq $16, %rsp

movl $90, -4(%rbp) //записываем 90 в degrees

pxor %xmm1, %xmm1 // очищаем регистр

cvtsi2sdl -4(%rbp), %xmm1 // приведение int к double

movq %xmm1, %rax

movq %rax, %xmm0

call toRadians

movq %xmm0, %rax // результат toRadians пишем в xmm0

movq %rax, %xmm0

call sinus

fstp %st(0) // Записать и вытолкнуть вещественное число (синус)

movl $0, %eax // return 0

leave

ret

.LC6:

.string "%ld \n"

main:

pushq %rbp

movq %rsp, %rbp

subq $48, %rsp

movl $90, -4(%rbp) // записываем 90 в degrees

leaq -32(%rbp), %rax // подготавливаем место для хранения start

movq %rax, %rsi

movl $4, %edi

call clock\_gettime // запоминаем начальное время

pxor %xmm3, %xmm3 // очищаем регистр

cvtsi2sdl -4(%rbp), %xmm3 // приведение int к double

movq %xmm3, %rax

movq %rax, %xmm0

call toRadians

movq %xmm0, %rax

movq %rax, %xmm0 // результат toRadians пишем в xmm0

call sinus

fstp %st(0) // Записать и вытолкнуть вещественное число (синус)

leaq -48(%rbp), %rax

movq %rax, %rsi

movl $4, %edi

call clock\_gettime // запоминаем время после выхода из функции синус

movq -48(%rbp), %rax

movq -32(%rbp), %rdx

subq %rdx, %rax // отнимаем start.tv от end.tv

pxor %xmm1, %xmm1

cvtsi2sdq %rax, %xmm1 // приводим результат к double

movq -40(%rbp), %rax

movq -24(%rbp), %rdx

subq %rdx, %rax //отнимаем наносекунды

pxor %xmm2, %xmm2

cvtsi2sdq %rax, %xmm2

movsd .LC5(%rip), %xmm0

mulsd %xmm2, %xmm0 //переводим наносекунда в секунды

addsd %xmm1, %xmm0 // складываем одно с другим

movsd %xmm0, -16(%rbp) //записываем полученный результат

movq -16(%rbp), %rax

movq %rax, %xmm0

movl $.LC6, %edi // передаем строку

movl $1, %eax // кол-во аргументов printf

call printf // печатаем

movl $0, %eax // return 0

leave

ret

.LC0:

.long -1577392366

.long 1066524486

.LC2:

.long 0

.long -1074790400

.LC3:

.long 0

.long 1073741824

.LC5:

.long -400107883

.long 1041313291

**Ассемблерный листинг для архитектуры x86-64 с оптимизацией O2**

toRadians:

mulsd .LC0(%rip), %xmm0 // частное можем вычислить на этапе компиляции, просто умножаем значение в регистре на него

ret

sinus:

pushq %r12

movl $1, %r12d // пишем единицы в f

pushq %rbp

movl $1, %ebp // и n

pushq %rbx

subq $48, %rsp

movsd %xmm0, 8(%rsp)

mulsd %xmm0, %xmm0

fldl 8(%rsp) // Помещаем в стек сопроцессора, т.е. st(0)

fldz // Push +0.0 onto the FPU register stack

movsd %xmm0, 32(%rsp)

.L2:

fnstcw 46(%rsp)

fadd %st(1), %st // складываем два последних числа на стеке

pxor %xmm1, %xmm1 // очистка регистра

cvtsi2sdl %ebp, %xmm1 // n приводится к инту

addl $1, %ebp

movzwl 46(%rsp), %eax // Copies the contents of the source operand (register or memory location) to the destination operand (register) and zero extends the value

fstpt 16(%rsp) // store 80-bit real and pop stack

orb $12, %ah // побитовое или с 1100

movw %ax, 44(%rsp)

fldcw 44(%rsp)

fistpl 8(%rsp)

fldcw 46(%rsp)

movl 8(%rsp), %ebx

movl %ebx, %eax

sarl $31, %eax

xorl %eax, %ebx

subl %eax, %ebx // всё что выше это округление

movq .LC1(%rip), %rax // записываем -1

movq %rax, %xmm0

call pow // -1 записываем в xmm0 чтобы передать первым аргуметом в pow

mulsd 32(%rsp), %xmm0 // Multiplies the low double-precision floating-point value

leal 1(%r12), %eax

addl $2, %r12d // прибавляем к f 2

imull %r12d, %eax // signed multiplication of two operands

movl %ebx, 40(%rsp)

fildl 40(%rsp) // Converts the signed-integer source operand into double extended-precision

// floating-point format and pushes the value onto the FPU register stack

cmpl $200000000, %ebp // проверка что n меньше N

movsd %xmm0, 8(%rsp)

fmull 8(%rsp)

movl %eax, 8(%rsp)

fildl 8(%rsp)

fdivrp %st, %st(1) // Divides the destination operand by the source operand and stores the result in the destination location.

fldt 16(%rsp) // загружаем на стек (в 16й лежит синус)

jne .L2 // если н меньше Н то след. итерация

fstp %st(1) // The FST instruction copies the value in the ST(0) register to the destination operand

addq $48, %rsp

popq %rbx // loads the value from the top of the stack to the location

popq %rbp

popq %r12

ret

.LC4:

.string "%ld \n"

main:

subq $40, %rsp

movl $4, %edi

movq %rsp, %rsi

call clock\_gettime

movl $90, %edi // записываем градусы

xorl %eax, %eax // чистим регистр

call toRadians

pxor %xmm0, %xmm0

cvtsi2sdl %eax, %xmm0 // degrees приводится к double

call sinus

fstp %st(0)

leaq 16(%rsp), %rsi

movl $4, %edi

call clock\_gettime

movq 24(%rsp), %rax

pxor %xmm0, %xmm0

subq 8(%rsp), %rax

cvtsi2sdq %rax, %xmm0

pxor %xmm1, %xmm1

movq 16(%rsp), %rax

subq (%rsp), %rax

mulsd .LC3(%rip), %xmm0 // перевод наносекунд

cvtsi2sdq %rax, %xmm1

movl $.LC4, %edi // передаем строку

movl $1, %eax // кол-во аргументов printf

addsd %xmm1, %xmm0 //складываем секунды

call printf

xorl %eax, %eax // обнуляем чтобы вернуть ноль

addq $40, %rsp

ret // // return 0

.LC1:

.long 0

.long -1074790400

.LC3:

.long -400107883

.long 1041313291

**Выводы**

По результатам проведённого анализа и сопоставления команд языка С и команд языка ассемблера было выяснено, что оптимизированном коде значительно уменьшено количество обращений в стек, больше используются регистры общего назначения и некоторые команды выполняются заранее. Код функции помещён в основное тело программы. Многие вычисления производятся с помощью специальных низкоуровневых алгоритмов работающих с битами.