###### 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 23201

Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Матвеев А.А.

Новосибирск 2024

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 <stdio.h>

#define N\_CONST 1000000000

double PiLeibniz(long long const n) {

double result = 0;

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

double const t = 1.0 / (2.0 \* (double)i + 1.0);

result = i % 2 == 0 ? result + t : result - t;

}

return result \* 4;

}

int main(void) {

double const result = PiLeibniz(N\_CONST);

printf("pi = %.8lf\n", result);

return 0;

}

**Ассемблерный листинг для архитектуры x86-64 без оптимизаций**

PiLeibniz:

push rbp ; сохраняем старое значение регистра RBP

mov rbp, rsp ; помещаем указатель стека RSP в регистр RBP

mov QWORD PTR [rbp-40], rdi ; записываем переменную n на стек со сдвигом 40; размер long long 8 байт

pxor xmm0, xmm0 ; запись 0 в регистр xmm0 (xor сам себя)

movsd QWORD PTR [rbp-8], xmm0 ; double result [rbp-8] = xmm0

mov QWORD PTR [rbp-16], 0 ; long long i [rbp-16] = 0

jmp .L2 ; переход на L2

.L5:

pxor xmm0, xmm0 ; зануление xmm0

cvtsi2sd xmm0, QWORD PTR [rbp-16] ; приведение long long i [rbp-16] к double и запись в xmm0

movapd xmm1, xmm0 ; xmm1 = xmm0

addsd xmm1, xmm0 ; xmm1 += xmm0 т.е. умножили на 2

movsd xmm0, QWORD PTR .LC1[rip] ; запись константы 1.0 в регистр xmm0

addsd xmm1, xmm0 ; xmm1 = xmm0

movsd xmm0, QWORD PTR .LC1[rip] ; запись константы 1.0 в регистр xmm0

divsd xmm0, xmm1 ; xmm0 /= xmm1

movsd QWORD PTR [rbp-24], xmm0 ; запись xmm0 в double t [rbp-24]

mov rax, QWORD PTR [rbp-16] ; запись i [rpb-16] в регистр rax

and eax, 1 ; eax принимает значение последнего бита, i нечет->1, чет->0

test rax, rax ; проверка значения rax

jne .L3 ; если не равно 0, переход на L3

movsd xmm0, QWORD PTR [rbp-8] ; иначе запись result [rbp-8] в регистр xmm0

addsd xmm0, QWORD PTR [rbp-24] ; xmm0 += t

jmp .L4 ; переход на L4

.L3:

movsd xmm0, QWORD PTR [rbp-8] ; запись result [rbp-8] в регистр xmm0

subsd xmm0, QWORD PTR [rbp-24] ; xmm0 -= t

.L4:

movsd QWORD PTR [rbp-8], xmm0 ; result [rbp-8] = xmm0

add QWORD PTR [rbp-16], 1 ; i += 1

.L2:

mov rax, QWORD PTR [rbp-16] ; запись i в rax

cmp rax, QWORD PTR [rbp-40] ; сравнение i, n

jl .L5 ; если i < n то переход к L5

movsd xmm1, QWORD PTR [rbp-8] ; иначе запись result [rbp-8] в регистр xmm1

movsd xmm0, QWORD PTR .LC2[rip] ; запись константы 4 в регистр xmm0

mulsd xmm0, xmm1 ; xmm0 \*= xmm1

pop rbp ; восстанавливаем в RBP значение для вызывающего кода

ret ; возвращаемся в вызывающий код

.LC3:

.string "pi = %.8lf\n"

main:

push rbp ; сохраняем старое значение регистра RBP

mov rbp, rsp ; помещаем указатель стека RSP в регистр RBP

sub rsp, 16 ; выделяем место в стеке под локальные переменные

mov edi, 1000000000 ; запись значения макроса N\_CONST в регистр edi для использования его функцией

call PiLeibniz ; помещаем адрес воврата в стек, вызываем функцию

movq rax, xmm0 ; запись всех битов из xmm0 в регистр rax

mov QWORD PTR [rbp-8], rax ; запись значения rax в result [rbp-8]

mov rax, QWORD PTR [rbp-8] ; запись значения result [rbp-8] в rax

movq xmm0, rax ; xmm0 = rax

mov edi, OFFSET FLAT:.LC3 ; запись константы-строки в edi

mov eax, 1 ; запись кол-ва дробных аргументов printf в eax

call printf ; печать строки

mov eax, 0 ; запись 0 в eax для возврата из main

leave ; <=> mov rsp, rbp; pop rbp

; удаляем локальные переменные и очищаем стек, восстанавливаем в RBP значение для вызывающего кода

ret ; возвращаемся в вызывающий код

.LC1:

.long 0

.long 1072693248

.LC2:

.long 0

.long 1074790400

**Ассемблерный листинг для архитектуры x86-64 с оптимизацией O1**

PiLeibniz:

test rdi, rdi ; проверка чему равен n [rdi]

jle .L6 ; если <= 0 то переход к L6, иначе идем дальше

mov eax, 0 ; i [eax] = 0

pxor xmm0, xmm0 ; зануление result [xmm0]

movsd xmm2, QWORD PTR .LC1[rip] ; запись константы 1.0 в xmm2

jmp .L5 ; переход к L5

.L3:

subsd xmm0, xmm3 ; result [xmm0] -= t [xmm3]

.L4:

add rax, 1 ; i += 1

cmp rdi, rax ; сравнение n [rdi], i [rax]

je .L2 ; если равны переход к L2, иначе к L5

.L5:

pxor xmm1, xmm1 ; зануление xmm1

cvtsi2sd xmm1, rax ; приведение i [rax] к double и запись в xmm1

addsd xmm1, xmm1 ; xmm1 \*= 2

addsd xmm1, xmm2 ; xmm1 += 1.0 [xmm2]

movapd xmm3, xmm2 ; t [xmm3] = 1.0 [xmm2]

divsd xmm3, xmm1 ; t [xmm3] /= xmm1

test al, 1 ; проверка последнего бита в i [rax]

jne .L3 ; если 1 (i нечет) то переход к L3

addsd xmm0, xmm3 ; иначе (i чет) result [xmm0] += t [xmm3]

jmp .L4 ; переход к L4

.L6:

pxor xmm0, xmm0 ; зануление result [xmm0]

.L2:

mulsd xmm0, QWORD PTR .LC2[rip] ; result [xmm0] \*= 4.0

ret ;

.LC3:

.string "pi = %.8lf\n"

main:

sub rsp, 8 ; выделяем место в стеке под локальные переменные

mov edi, 1000000000 ; запись значения макроса N\_CONST в регистр edi для использования его функцией

call PiLeibniz ; помещаем адрес воврата в стек, вызываем функцию

mov edi, OFFSET FLAT:.LC3 ; запись константы-строки в edi

mov eax, 1 ; запись кол-ва дробных аргументов printf в eax

call printf ; печать строки

mov eax, 0 ; запись 0 в eax для возврата из main

add rsp, 8 ; освобождаем стек

ret ; возвращаемся в вызывающий код

.LC1:

.long 0

.long 1072693248

.LC2:

.long 0

.long 1074790400

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

PiLeibniz:

test rdi, rdi ; проверка чему равен n [rdi]

jle .L6 ; если <= 0 то переход к L6, иначе идем дальше

movsd xmm2, QWORD PTR .LC1[rip] ; запись константы 1.0 в xmm2

xor eax, eax ; зануление i [eax]

pxor xmm0, xmm0 ; зануление result [xmm0]

jmp .L5 ; переход к L5

.L10:

add rax, 1 ; i [rax] += 1

addsd xmm0, xmm3 ; result [xmm0] += t [xmm3]

cmp rdi, rax ; сравнение n [rdi], i [rax]

je .L9 ; если равны то переход к L9, иначе к L5

.L5:

pxor xmm1, xmm1 ; зануление xmm1

movapd xmm3, xmm2 ; t [xmm3] = 1.0 [xmm2]

cvtsi2sd xmm1, rax ; xmm1 = (double)i [rax]

addsd xmm1, xmm1 ; xmm1 \*= 2

addsd xmm1, xmm2 ; xmm1 += 1.0 [xmm2]

divsd xmm3, xmm1 ; t [xmm3] /= xmm1

test al, 1 ; проверка последнего бита в i [rax]

je .L10 ; если 0 (i чет) то переход к L10

add rax, 1 ; иначе i [rax] += 1

subsd xmm0, xmm3 ; result [xmm0] -= t [xmm3]

cmp rdi, rax ; сравнение n [rdi], i [rax]

jne .L5 ; если не равны (т.е. i<n) то переход к L5, иначе к L9

.L9:

mulsd xmm0, QWORD PTR .LC2[rip] ; result [xmm0] \*= 4

ret ;

.L6:

pxor xmm0, xmm0 ; зануление result [xmm0]

ret ;

.LC3:

.string "pi = %.8lf\n"

main:

sub rsp, 8

mov edi, 1000000000

call PiLeibniz

mov edi, OFFSET FLAT:.LC3

mov eax, 1

call printf

xor eax, eax

add rsp, 8

ret

.LC1:

.long 0

.long 1072693248

.LC2:

.long 0

.long 1074790400

**Ассемблерный листинг для архитектуры x86-64 с оптимизацией O3, Ofast**

PiLeibniz:

test rdi, rdi

jle .L6

movsd xmm2, QWORD PTR .LC1[rip]

xor eax, eax

pxor xmm0, xmm0

jmp .L5

.L10:

add rax, 1

addsd xmm0, xmm3

cmp rdi, rax

je .L9

.L5:

pxor xmm1, xmm1

movapd xmm3, xmm2

cvtsi2sd xmm1, rax

addsd xmm1, xmm1

addsd xmm1, xmm2

divsd xmm3, xmm1

test al, 1

je .L10

add rax, 1

subsd xmm0, xmm3

cmp rdi, rax

jne .L5

.L9:

mulsd xmm0, QWORD PTR .LC2[rip]

ret

.L6:

pxor xmm0, xmm0

ret

.LC3:

.string "pi = %.8lf\n"

main:

sub rsp, 8

movsd xmm2, QWORD PTR .LC1[rip]

pxor xmm0, xmm0

xor eax, eax

.L15:

pxor xmm1, xmm1

movapd xmm3, xmm2

cvtsi2sd xmm1, rax

addsd xmm1, xmm1

addsd xmm1, xmm2

divsd xmm3, xmm1

test al, 1

jne .L12

addsd xmm0, xmm3

add rax, 1

jmp .L15

.L12:

add rax, 1

subsd xmm0, xmm3

cmp rax, 1000000000

jne .L15

mulsd xmm0, QWORD PTR .LC2[rip]

mov edi, OFFSET FLAT:.LC3

mov eax, 1

call printf

xor eax, eax

add rsp, 8

ret

.LC1:

.long 0

.long 1072693248

.LC2:

.long 0

.long 1074790400

**Выводы**

По результатам проведённого анализа и сопоставления команд языка С и команд языка ассемблера было выяснено, что в оптимизированном коде значительно уменьшено количество обращений в стек, больше используются регистры общего назначения и некоторые команды выполняются заранее. На уровне O3 Код функции помещён в основное тело программы. Многие вычисления производятся с помощью специальных низкоуровневых алгоритмов работающих с битами.