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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Введение в архитектуру x86/x86-64»

студента *2* курса, группы 21204

Осипова Александра Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
А. Ю. Власенко

Новосибирск 2022

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Знакомство с программной архитектурой x86-64.
2. Анализ ассемблерного листинга программы для архитектуры x86-64.

ЗАДАНИЕ

1. Изучить программную архитектуру x86-64.
2. Для программы на языке Си (вычисление числа Пи с помощью первых N членов ряда Грегори-Лейбница) сгенерировать ассемблерные листинги для архитектуры x86 или x86-64, используя уровни оптимизации O0 и O3.
3. Проанализировать полученные листинги и сделать следующее:

- сопоставить команды языка Си с машинными командами;
- определить размещение переменных языка Си в программах на ассемблере;
- выписать оптимизационные преобразования, выполненные компилятором;

4. Составить отчет по лабораторной работе.

ОПИСАНИЕ РАБОТЫ

1. С помощью онлайн сервиса godbolt.org, для архитектуры x86-64 был сгенерирован ассемблерный листинг программы, написанной на языке Си (см. Приложение 1), для вычисления числа Пи с помощью первых N членов ряда Грегори-Лейбница, используя уровни оптимизации O0 и O3.

2. Были проанализированы полученные листинги и сопоставлены команды языка Си с машинными командами (см. Приложение 2, 3).

3. Оптимизационные преобразования, сделанные компилятором:

3.1. В случае, когда N=1, программа сразу присваивает переменной pi значение 4 и возвращает его, минуя цикл.

3.2. Перед выходом из функции, программа удаляет тот стековый кадр, который выделила в начале функции.

```
subq    $8, %rsp
        ...
addq    $8, %rsp
ret
```

3.3. Уменьшалось общее количество строк в листинге.

3.4. Компилятор сам применил команду inline для функции double PiByGregoryLeibniz(long long int N), вставив ее в тело функции void TestClockGettime(long long int N). Тем самым программа не тратит ресурсы компьютера для создания дополнительного стекового кадра.

3.5. Процессор стал больше работать с регистрами, а не с ячейками оперативной памяти. За счет этого достигается ускорение в работе программы.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы были сгенерированы и проанализированы ассемблерные листинги с уровнями оптимизации O0 и O3 на архитектуре x86-64 для программы на Си, реализующей вычисление числа Пи с помощью ряда Грегори-Лейбница. Были зафиксированы такие оптимизационные преобразования, как удаление стекового кадра, inline функции PiByGregoryLeibniz, условие, при котором в функции PiByGregoryLeibniz минуется цикл (N==1). Так же основное преобразование заключается в том, что процессор стал больше работать с регистрами, а не с оперативной памятью.

Приложение 1. Исходный код программы на Си

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

double PiByGregoryLeibniz(long long int N){
    double pi = 1.0;
    double pow = -1;
    for (long long int i = 1; i <= N; ++i){
        pi += pow/(2*i+1);
        pow *= (-1);
    }
    pi *= 4;
    return pi;
}

void TestClockGettime(long long int N){
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start);
    double pi = PiByGregoryLeibniz(N);
    clock_gettime(CLOCK_MONOTONIC_RAW, &end);
    printf("Pi: %f\n", pi);
    printf("Time taken by clock_gettime: %lf sec.\n", \
    end.tv_sec - start.tv_sec + 0.000000001*(end.tv_nsec - start.tv_nsec));
}

int main(int argc, char** argv){
    long long int N = 0;
    N = atoll(argv[1]);
    TestClockGettime(N);
    return 0;
}
```

Приложение 2. Анализ ассемблерного листинга с уровнем оптимизации O0

```
PiByGregoryLeibniz: //PiByGregoryLeibniz (до цикла)
        pushq   %rbp                    //создаем стековый кадр
        movq    %rsp, %rbp
        movq    %rdi, -40(%rbp)         //Кладем на стек аргумент N
        movsd   .LC0(%rip), %xmm0       //double pi = 1.0
        movsd   %xmm0, -8(%rbp)         //pi: -8(%rbp)
        movsd   .LC1(%rip), %xmm0       //double pow = -1
        movsd   %xmm0, -16(%rbp)        //pow: -16(%rbp)
        movq    $1, -24(%rbp)           //i = 1: -24(%rbp)
        jmp     .L2                     //Переходим к циклу
```

```
.L3: //Здесь реализовано тело цикла
//Этот блок реализует pi += pow/(2*i + 1)
        movq    -24(%rbp), %rax
        addq    %rax, %rax
        addq    $1, %rax
        pxor    %xmm1, %xmm1
        cvtsi2sdq       %rax, %xmm1
        movsd   -16(%rbp), %xmm0
        divsd   %xmm1, %xmm0
        movsd   -8(%rbp), %xmm1
        addsd   %xmm1, %xmm0
//Этот блок реализует pow *= (-1)
        movsd   %xmm0, -8(%rbp)
        movsd   -16(%rbp), %xmm0
        movq    .LC2(%rip), %xmm1
        xorpd   %xmm1, %xmm0
        movsd   %xmm0, -16(%rbp)
//++i
        addq    $1, -24(%rbp)

.L2: //Условие цикла for и конец функции PiByGregoryLeibniz
//Условие цикла for
        movq    -24(%rbp), %rax
        cmpq    -40(%rbp), %rax
        jle     .L3    //в случае истинности cmpq переходим к телу цикла
//Этот блок реализует pi *= 4
        movsd   -8(%rbp), %xmm1
        movsd   .LC3(%rip), %xmm0
        mulsd   %xmm1, %xmm0
        movsd   %xmm0, -8(%rbp)
//Этот блок реализует выход из функции PiByGregoryLeibniz
        movsd   -8(%rbp), %xmm0
        movq    %xmm0, %rax
        movq    %rax, %xmm0
        popq    %rbp
        ret

.LC4:
        .string "Pi: %f\n"
.LC6:
        .string "Time taken by clock_gettime: %lf sec.\n"

TestClockGettime: //Реализация функции TestClockGettime
//Создание стекового кадра
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $64, %rsp
        movq    %rdi, -56(%rbp)        //N: -56(%rbp)
```

```
//Этот блок реализует вызов clock_gettime для start
        leaq    -32(%rbp), %rax        //start: -32(%rbp)
        movq    %rax, %rsi
        movl    $4, %edi
        call    clock_gettime
//Этот блок отвечает за вызов функции PiByGregoryLeibniz
        movq    -56(%rbp), %rax
        movq    %rax, %rdi
        call    PiByGregoryLeibniz
        movq    %xmm0, %rax
        movq    %rax, -8(%rbp)         //pi: -8(%rbp)
//Этот блок реализует вызов clock_gettime для end
        leaq    -48(%rbp), %rax        //end: -48(%rbp)
        movq    %rax, %rsi
        movl    $4, %edi
        call    clock_gettime
//Этот блок отвечает за вызов printf для pi
        movq    -8(%rbp), %rax
        movq    %rax, %xmm0
        movl    $.LC4, %edi
        movl    $1, %eax
        call    printf
//Этот блок отвечает за вызов printf для вывода времени
// Реализуем end.tv_sec - start.tv_sec
        movq    -48(%rbp), %rdx
        movq    -32(%rbp), %rax
        subq    %rax, %rdx

        pxor    %xmm1, %xmm1
        cvtsi2sdq       %rdx, %xmm1
// Реализуем end.tv_nsec - start.tv_nsec
        movq    -40(%rbp), %rdx
        movq    -24(%rbp), %rax
        subq    %rax, %rdx
        pxor    %xmm2, %xmm2
        cvtsi2sdq       %rdx, %xmm2
//Реализуем 0.000000001*(end.tv_nsec - start.tv_nsec)
        movsd   .LC5(%rip), %xmm0
        mulsd   %xmm2, %xmm0
//Реализуем end.tv_sec - start.tv_sec + 0.000000001*(end.tv_nsec -
start.tv_nsec)
        addsd   %xmm0, %xmm1
        movq    %xmm1, %rax
//Вызываем printf
        movq    %rax, %xmm0
        movl    $.LC6, %edi
        movl    $1, %eax
        call    printf
```

```
//Выходим из функции
        nop
        leave
        ret

main:
//Выделяем стековый кадр
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
//Кладем аргументы main в стек
        movl    %edi, -20(%rbp)
        movq    %rsi, -32(%rbp)

        movq    $0, -8(%rbp)            //N = 0: -8(%rbp)
//Этот блок отвечает за вызов atoll
        movq    -32(%rbp), %rax
        addq    $8, %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        call    atoll
        movq    %rax, -8(%rbp)          //N: -8(%rbp)
//Этот блок отвечает за вызов подпрограммы ClockGettime
        movq    -8(%rbp), %rax
        movq    %rax, %rdi
        call    TestClockGettime
//Выходим из функции
        movl    $0, %eax
        leave
        ret

.LC0: //double 1
        .long   0
        .long   1072693248

.LC1: //double -1
        .long   0
        .long   -1074790400

.LC2: //double -1
        .long   0
        .long   -2147483648
        .long   0
        .long   0

.LC3: //double 4
        .long   0
        .long   1074790400
```

```
.LC5: //double 0.000000001
        .long   -400107883
        .long   1041313291
```

Приложение 3. Анализ ассемблерного листинга с уровнем оптимизации O3

```
PiByGregoryLeibniz:
        testq   %rdi, %rdi              //N: %rdi
        jle     .L4                     //если N==1, то переходим по .L4
        movsd   .LC0(%rip), %xmm1       //pow = -1: %xmm1
        leaq    3(%rdi,%rdi), %rdx      //N: %rdx
        movl    $3, %eax
        movsd   .LC1(%rip), %xmm0       //pi = 1.0: %xmm0
        movq    .LC3(%rip), %xmm4
.L3:
// В этом блоке реализован цикл в функции PiByGregoryLeibniz
        pxor    %xmm3, %xmm3            //(%xmm3) = 0
        movapd  %xmm1, %xmm2            //pow: %xmm2
        xorpd   %xmm4, %xmm1
        cvtsi2sdq       %rax, %xmm3
        addq    $2, %rax
        divsd   %xmm3, %xmm2            //pow /= (2*i+1)
        addsd   %xmm2, %xmm0            //pi += pow/(2*i+1)
        cmpq    %rax, %rdx
        jne     .L3
//Выход из функции
        mulsd   .LC2(%rip), %xmm0
        ret
.L4:
        movsd   .LC2(%rip), %xmm0
        ret


.LC4:
        .string "Pi: %f\n"
.LC6:
        .string "Time taken by clock_gettime: %lf sec.\n"

TestClockGettime:
        pushq   %rbx
        movq    %rdi, %rbx              //N: %rbx
        movl    $4, %edi
        subq    $48, %rsp
        leaq    16(%rsp), %rsi          //start: 16(%rsp)
        call    clock_gettime
        testq   %rbx, %rbx
        jle     .L10                    //если N==1, то переходим в .L10
```

```
        //Ниже представлен тот же набор команд, что и функции
        //PiByGregoryLeibniz
         movsd   .LC1(%rip), %xmm0
         leaq    3(%rbx,%rbx), %rdx
         movl    $3, %eax
         movsd   .LC0(%rip), %xmm1
         movq    .LC3(%rip), %xmm4

.L9:  //Цикл, аналогичный циклу в PiByGregoryLeibniz
         pxor    %xmm3, %xmm3
         movapd  %xmm1, %xmm2
         xorpd   %xmm4, %xmm1
         cvtsi2sdq       %rax, %xmm3
         addq    $2, %rax
         divsd   %xmm3, %xmm2
         addsd   %xmm2, %xmm0
         cmpq    %rax, %rdx
         jne     .L9
//Выход из цикла
         mulsd   .LC2(%rip), %xmm0

.L8:
//Этот блок отвечает за вызов функции clock_gettime
         leaq    32(%rsp), %rsi          //end: 32(%rsp)
         movl    $4, %edi
         movsd   %xmm0, 8(%rsp)
         call    clock_gettime
//Работа команды printf для Pi
         movsd   8(%rsp), %xmm0          //pi: %xmm0
         movl    $.LC4, %edi
         movl    $1, %eax
         call    printf
//Реализуем (end.tv_nsec - start.tv_nsec)
         movq    40(%rsp), %rax          //end.tv_nsec: 40(%rsp)
         pxor    %xmm0, %xmm0
         subq    24(%rsp), %rax          //start.tv_nsec: 24(%rsp)
         cvtsi2sdq       %rax, %xmm0
//Реализуем (end.tv_sec - start.tv_sec)
         pxor    %xmm1, %xmm1
         movq    32(%rsp), %rax          //end.tv_sec: 32(%rsp)
         subq    16(%rsp), %rax          //start.tv_sec: 16(%rsp)
//Реализуем 0.000000001*(end.tv_nsec - start.tv_nsec)
         mulsd   .LC5(%rip), %xmm0
//Реализуем end.tv_sec - start.tv_sec + 0.000000001*(end.tv_nsec - start.tv_nsec)
         cvtsi2sdq     %rax, %xmm1
         movl    $.LC6, %edi
         movl    $1, %eax
         addsd   %xmm1, %xmm0
```

```
        call    printf
//Выходим из функции TestClockGettime
        addq    $48, %rsp
        popq    %rbx
        ret

.L10: //Условный переход, если N==1
        movsd   .LC2(%rip), %xmm0       //pi = 4: %xmm0
        jmp     .L8

main:
        subq    $8, %rsp
//Преобразование строки в long long int
        movq    8(%rsi), %rdi
        movl    $10, %edx
        xorl    %esi, %esi
        call    strtoll
        movq    %rax, %rdi

        call    TestClockGettime

        xorl    %eax, %eax
        addq    $8, %rsp
        ret


.LC0: double -1
        .long   0
        .long   -1074790400

.LC1: //double 1
        .long   0
        .long   1072693248

.LC2: //double 4
        .long   0
        .long   1074790400

.LC3:
        .long   0
        .long   -2147483648
        .long   0
        .long   0

.LC5: //double 0.000000001
        .long   -400107883
        .long   1041313291
```