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

«Знакомство с программной архитектурой x86/x86-64 и анализ ассемблерного листинга»

студента 2 курса, группы 20203

Синюкова Валерия Константиновича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент кафедры параллельных вычислений
Власенко Андрей Юрьевич

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Знакомство с программной архитектурой x86/x86-64.

2. Анализ ассемблерного листинга программы для архитектуры x86/x86-64.

ЗАДАНИЕ

Формулировка общего задания:

1. Изучить программную архитектуру x86/x86-64:

· набор регистров,

· основные арифметико-логические команды,

· способы адресации памяти,

· способы передачи управления,

· работу со стеком,

· вызов подпрограмм,

· передачу параметров в подпрограммы и возврат результатов,

· работу с арифметическим сопроцессором,

· работу с векторными расширениями.

2. Для программы на языке Си (вычисление числа Пи с помощью разложения в ряд Грегори-Лейбница) сгенерировать ассемблерные листинги (синтаксис AT&T, принятый в UNIX) для архитектуры x86 или архитектуры 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 (в каких регистрах, в каких ячейках памяти);

· выписать оптимизационные преобразования, выполненные компилятором;

4. Составить отчет по лабораторной работе. Отчет должен содержать следующее:

· Титульный лист.

· Цель лабораторной работы.

· Полный компилируемый листинг реализованной программы на Си.

· Листинги на ассемблере (O0 и O3).

· Вывод по результатам лабораторной работы.

ОПИСАНИЕ РАБОТЫ

1) Были сгенерированы ассемблерные листинги для программы на языке C «вычисление числа Пи с помощью разложения в ряд Грегори-Лейбница» для архитектуры x86-64 с уровнями компиляции O0 и O3.
2) Были сопоставлены команды программы на языке C с командами получившихся ассемблерных листингов. Далее приведен полный компилируемый листинг программы на C и ассемблерный листинг программы с уровнем компиляции O0 с комментариями, каким командам на языке C соответствуют различные секции ассемблерного кода.

Код на языке C:

```
#include <stdio.h>
#include <string.h>

double LeibnizFormula (long long N)
{
   double Pi = 0, nextMember;
   long long i;
   for (i = 0; i < N; i++)
   {
        if (i % 2)
           nextMember = -1;
        else
           nextMember = 1;
        nextMember /= 2*(double)(i) + 1;
        Pi += nextMember;
   }
   Pi *= 4;
   return Pi;
}

int fromCharToInt (char c)
{
   return (int)(c - '0');
}
```

```
long long getN (char * charN)
{
   int i, lengthN = strlen(charN);
   long long N = 0;
   for (i = 0; i < lengthN; i++)
   {
        N *= 10;
        N += (long long)(fromCharToInt(charN[i]));
   }
   return N;
}

void printAnswer(double Pi)
{
   printf("%f\n",Pi);
}

int main(int argc, char** argv)
{
   long long N = getN(argv[1]);
   double Pi = LeibnizFormula(N);
   printAnswer(Pi);
   return 0;
}
```

Ассемблерный код с уровнем оптимизации O0:

```
        .file   "lab2.c"
        .text                                   // секция кода
        .globl LeibnizFormula
        .type  LeibnizFormula, @function
LeibnizFormula:
.LFB0:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6                 // пролог
        movq %rdi, -40(%rbp)                    // создание локальной копии N,
                                                   помещение ее в стек по адресу
                                                   -40(%rbp)
        pxor   %xmm0, %xmm0
        movsd          %xmm0, -24(%rbp)         // -24(%rbp) = Pi
        movq $0, -8(%rbp)                       // -8(%rbp) = i = 0
```

```
        jmp    .L2
.L5:
        movq -8(%rbp), %rax
        andl   $1, %eax
        testq  %rax, %rax
        je     .L3                              // если !(i % 2)
        movsd       .LC1(%rip), %xmm0
        movsd       %xmm0, -16(%rbp)        // Pi = -1
        jmp    .L4
.L3:
        movsd       .LC2(%rip), %xmm0
        movsd       %xmm0, -16(%rbp)        // Pi = 1
.L4:
        pxor   %xmm0, %xmm0                 /*
        cvtsi2sdq     -8(%rbp), %xmm0
        addsd %xmm0, %xmm0
        movsd       .LC2(%rip), %xmm1
        addsd %xmm1, %xmm0                    nextMember /= 2*(double)(i) + 1
        movsd       -16(%rbp), %xmm1
        divsd  %xmm0, %xmm1
        movapd      %xmm1, %xmm0
        movsd       %xmm0, -16(%rbp)                                    */
        movsd       -24(%rbp), %xmm0        /*
        addsd -16(%rbp), %xmm0                      Pi += nextMember
        movsd       %xmm0, -24(%rbp)                                    */
        addq   $1, -8(%rbp)                 // i++
.L2:
        movq -8(%rbp), %rax
        cmpq -40(%rbp), %rax
        jl     .L5                              // если i < N, то переходим на
                                                  следующую итерацию цикла
        movsd       -24(%rbp), %xmm1        /*
        movsd       .LC3(%rip), %xmm0
        mulsd%xmm1, %xmm0                           Pi *= 4;
        movsd       %xmm0, -24(%rbp)
        movsd       -24(%rbp), %xmm0                                    */
        popq   %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc                        // эпилог
.LFE0:
        .size   LeibnizFormula, .-LeibnizFormula
        .globl fromCharToInt
        .type   fromCharToInt, @function
fromCharToInt:
```

```
.LFB1:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6                 // пролог
        movl  %edi, %eax
        movb %al, -4(%rbp)                      // создание локальной копии c =
                                                   charN[i], помещение ее в стек по
                                                   адресу -4(%rbp)
        movsbl      -4(%rbp), %eax
        subl   $48, %eax                        // %eax = c - '0'
        popq  %rbp                              // эпилог
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   fromCharToInt, .-fromCharToInt
        .globl getN
        .type  getN, @function
getN:
.LFB2:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6                 // пролог
        subq  $32, %rsp
        movq %rdi, -24(%rbp)                    // создание локальной копии charN
                                                   = argv[1], помещение ее в стек по
                                                   адресу -24(%rbp)
        movq -24(%rbp), %rax
        movq %rax, %rdi
        call    strlen
        movl  %eax, -12(%rbp)                   // -12(%rbp) = lengthN = strlen(charN)
        movq $0, -8(%rbp)                       // -8(%rbp) = N = 0
        movl  $0, -16(%rbp)                     // -16(%rbp) = i = 0
        jmp    .L10
.L11:
        movq -8(%rbp), %rdx                     /*
        movq %rdx, %rax
        salq   $2, %rax                                 N *= 10
        addq  %rdx, %rax
```

```
        addq  %rax, %rax
        movq %rax, -8(%rbp)                                                 */

        movl  -16(%rbp), %eax           // %eax = i
        movslq        %eax, %rdx
        movq -24(%rbp), %rax            // %rax = charN
        addq  %rdx, %rax
        movzbl        (%rax), %eax      // %al = charN[i]
        movsbl        %al, %eax
        movl  %eax, %edi                // %edi = charN[i]
        call    fromCharToInt
        cltq
        addq  %rax, -8(%rbp)            // N += fromCharToInt(charN[i])
        addl   $1, -16(%rbp)            // i++
.L10:
        movl  -16(%rbp), %eax
        cmpl  -12(%rbp), %eax
        jl        .L11                  // если lengthN > i, то переходим на
                                           следующую итерацию цикла for
        movq -8(%rbp), %rax             // полностью перевели параметр N
                                           из последовательности символов в
                                           int
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc                    // эпилог
.LFE2:
        .size   getN, .-getN
        .section        .rodata
.LC4:
        .string"%f\n"
        .text
        .globl printAnswer
        .type  printAnswer, @function
printAnswer:
.LFB3:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6         // пролог
        subq   $16, %rsp
        movsd         %xmm0, -8(%rbp)   // -8(%rbp) = Pi
        movq -8(%rbp), %rax
```

```
        movq %rax, -16(%rbp)                // -16(%rbp) = Pi
        movsd       -16(%rbp), %xmm0
        movl  $.LC4, %edi
        movl  $1, %eax
        call    printf
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE3:
        .size   printAnswer, .-printAnswer
        .globl main
        .type  main, @function
main:
.LFB4:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6             // пролог
        subq  $48, %rsp
        movl  %edi, -20(%rbp)               // -20(%rbp) = argc
        movq %rsi, -32(%rbp)                // -32(%rbp) = argv
        movq -32(%rbp), %rax
        addq  $8, %rax
        movq (%rax), %rax
        movq %rax, %rdi                     // %rdi = argv[1]
        call    getN
        movq %rax, -16(%rbp)                // -16(%rbp) = N;
        movq -16(%rbp), %rax
        movq %rax, %rdi
        call    LeibnizFormula
        movq %xmm0, %rax
        movq %rax, -8(%rbp)                 // -8(%rbp) = Pi;
        movq -8(%rbp), %rax
        movq %rax, -40(%rbp)                // -40(%rbp) = Pi
        movsd       -40(%rbp), %xmm0
        call    printAnswer
        movl  $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

```
.LFE4:
        .size   main, .-main
        .section        .rodata
        .align 8
.LC1:
        .long  0
        .long  -1074790400
        .align 8
.LC2:
        .long  0
        .long  1072693248
        .align 8
.LC3:
        .long  0
        .long  1074790400
        .ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"
        .section        .note.GNU-stack,"",@progbits
```

3) Были выделены оптимизационные преобразования, выполненные при компиляции с уровнем оптимизации O3 (сам листинг приведен после списка выделенных оптимизаций):

1. Значения почти всех переменных хранятся в регистрах процессора, а не в стеке. Данные, необходимые для выполнения нашей программы, не занимают место в ОП, не тратится время на чтение данных из стека и на запись данных в стек.
2. В функции main не создаются переменные argc и argv, так как они никак не влияют на результат работы программы.
3. Было произведено встраивание тел функций fromCharToInt, printAnswer и LeibnizFormula в функцию main, для устранения расходов времени на вызов функций и передачу в них аргументов. При этом функции fromCharToInt и printAnswer даже не были преобразованы в ассемблерный код.
4. Соблюдается выравнивание при помощи команды .p2align для более быстрого доступа к данным.
5. Был оптимизирован данный сегмент кода из функции getN:
   ```
   N *= 10;
   N += (long long)(fromCharToInt(charN[i]));
   ```
   При уровне оптимизации O0 данный сегмент был преобразован в ассемблерный код следующим образом (*см. метку .L11 соответствующего листинга*):
   - Пусть изначальное значение – N.
   - Создается копия данного значение: $N_1 = N$.
   - Выполняется побитовый сдвиг: $N = N << 2$.
   - Затем прибавляется копия: $N = N + N_1$.

- Затем выполняется умножение N на 2 с помощью операции сложения: N = N + N.
- Лишь затем, после вычисления (long long)(fromCharToInt(charN[i])) выполняется N += (long long)(fromCharToInt(charN[i])).

При уровне оптимизации O3 данный сегмент был преобразован следующим образом (*см. метку .L14 соответствующего листинга*):

- С помощью команды lea выполняется N *= 5: **leaq (%rax,%rax,4), %rcx** (в регистре %rax находится начальное значение N).
- Затем вычисляется значение (long long)(fromCharToInt(charN[i]).
- Затем с помощью команды lea вычисляется N = 2*N + (long long)(fromCharToInt(charN[i]) : **leaq (%rax,%rcx,2), %rax**.

6. В случае, если в функции getN (lengthN == 0), сразу происходит выход из функции (*см. метку .L15 ассемблерного листинга с уровнем оптимизации O3*).

Ассемблерный код с уровнем оптимизации O3:

```
.file   "lab2.c"
        .section        .text.unlikely,"ax",@progbits
.LCOLDB4:
        .text
.LHOTB4:
        .p2align 4,,15
        .globl LeibnizFormula
        .type  LeibnizFormula, @function
LeibnizFormula:
.LFB47:
        .cfi_startproc
        testq   %rdi, %rdi
        jle     .L6
        movsd       .LC1(%rip), %xmm3
        xorl    %eax, %eax
        pxor    %xmm0, %xmm0
        movapd      %xmm3, %xmm2
        movapd      %xmm3, %xmm4
        movsd       .LC2(%rip), %xmm5
        .p2align 4,,10
        .p2align 3
.L3:
        pxor    %xmm1, %xmm1
        cvtsi2sdq     %rax, %xmm1
        addq    $1, %rax
        cmpq %rax, %rdi
        addsd %xmm1, %xmm1
```

```
        addsd %xmm3, %xmm1
        divsd  %xmm1, %xmm2
        addsd %xmm2, %xmm0
        je      .L10
        testb   $1, %al
        movapd      %xmm4, %xmm2
        je      .L3
        movapd      %xmm5, %xmm2
        jmp    .L3
        .p2align 4,,10
        .p2align 3
.L10:
        mulsd .LC3(%rip), %xmm0
        ret
.L6:
        pxor   %xmm0, %xmm0
        ret
        .cfi_endproc
.LFE47:
        .size   LeibnizFormula, .-LeibnizFormula
        .section        .text.unlikely
.LCOLDE4:
        .text
.LHOTE4:
        .section        .text.unlikely
.LCOLDB5:
        .text
.LHOTB5:
        .p2align 4,,15
        .globl fromCharToInt
        .type  fromCharToInt, @function
fromCharToInt:
.LFB48:
        .cfi_startproc
        movsbl      %dil, %eax
        subl    $48, %eax
        ret
        .cfi_endproc
.LFE48:
        .size   fromCharToInt, .-fromCharToInt
        .section        .text.unlikely
.LCOLDE5:
        .text
.LHOTE5:
        .section        .text.unlikely
```

```
.LCOLDB6:
        .text
.LHOTB6:
        .p2align 4,,15
        .globl getN
        .type  getN, @function
getN:
.LFB49:
        .cfi_startproc
        pushq %rbx
        .cfi_def_cfa_offset 16
        .cfi_offset 3, -16
        movq %rdi, %rbx
        call    strlen
        testl   %eax, %eax
        jle     .L15
        movl  %eax, %esi
        xorl    %edx, %edx
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L14:
        leaq    (%rax,%rax,4), %rcx
        movsbl        (%rbx,%rdx), %eax
        addq  $1, %rdx
        subl    $48, %eax
        cmpl  %edx, %esi
        cltq
        leaq    (%rax,%rcx,2), %rax
        jg      .L14
        popq  %rbx
        .cfi_remember_state
        .cfi_def_cfa_offset 8
        ret
.L15:
        .cfi_restore_state
        xorl    %eax, %eax
        popq  %rbx
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE49:
        .size   getN, .-getN
        .section        .text.unlikely
.LCOLDE6:
```

```
        .text
.LHOTE6:
        .section        .rodata.str1.1,"aMS",@progbits,1
.LC7:
        .string"%f\n"
        .section        .text.unlikely
.LCOLDB8:
        .text
.LHOTB8:
        .p2align 4,,15
        .globl printAnswer
        .type  printAnswer, @function
printAnswer:
.LFB50:
        .cfi_startproc
        movl  $.LC7, %esi
        movl  $1, %edi
        movl  $1, %eax
        jmp    __printf_chk
        .cfi_endproc
.LFE50:
        .size   printAnswer, .-printAnswer
        .section        .text.unlikely
.LCOLDE8:
        .text
.LHOTE8:
        .section        .text.unlikely
.LCOLDB9:
        .section        .text.startup,"ax",@progbits
.LHOTB9:
        .p2align 4,,15
        .globl main
        .type  main, @function
main:
.LFB51:
        .cfi_startproc
        subq   $8, %rsp
        .cfi_def_cfa_offset 16
        movq 8(%rsi), %rdi
        call    getN
        testq   %rax, %rax
        pxor   %xmm0, %xmm0
        jle      .L20
        movsd         .LC1(%rip), %xmm3
        xorl    %edx, %edx
```

```
        pxor   %xmm0, %xmm0
        movapd      %xmm3, %xmm2
        movsd       .LC2(%rip), %xmm4
        movapd      %xmm3, %xmm5
        .p2align 4,,10
        .p2align 3
.L21:
        pxor   %xmm1, %xmm1
        cvtsi2sdq     %rdx, %xmm1
        addq   $1, %rdx
        cmpq %rdx, %rax
        addsd %xmm1, %xmm1
        addsd %xmm3, %xmm1
        divsd  %xmm1, %xmm2
        addsd %xmm2, %xmm0
        je      .L20
        testb   $1, %dl
        movapd      %xmm4, %xmm2
        jne     .L21
        movapd      %xmm5, %xmm2
        jmp    .L21
        .p2align 4,,10
        .p2align 3
.L20:
        mulsd .LC3(%rip), %xmm0
        movl   $.LC7, %esi
        movl   $1, %edi
        movl   $1, %eax
        call    __printf_chk
        xorl    %eax, %eax
        addq   $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE51:
        .size   main, .-main
        .section        .text.unlikely
.LCOLDE9:
        .section        .text.startup
.LHOTE9:
        .section        .rodata.cst8,"aM",@progbits,8
        .align 8
.LC1:
        .long  0
        .long  1072693248
```

```
        .align 8
.LC2:
        .long  0
        .long  -1074790400
        .align 8
.LC3:
        .long  0
        .long  1074790400
        .ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"
        .section        .note.GNU-stack,"",@progbits
```

ЗАКЛЮЧЕНИЕ

В результате данной практической работы была изучена программная архитектура x86-64, были сгенерированы и проанализированы ассемблерные листинги программы на языке С с уровнями оптимизации O0 и O3. Ассемблерный код программы был сопоставлен с кодом на языке С. Были выделены преобразования, такие как удаление мертвого кода, встраивание функций, отображение переменных на регистры процессора и т.д., выполненные компилятором с уровнем оптимизации O3 для сокращения затрат времени на работу программы и уменьшения размеров памяти, занимаемой программой.