###### 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

«Лабораторная работа 3»

Студента 1 курса, 19210 группы

**Пирожков Андрей Константинович**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

(ученая степень, звание)

Д.С.Иванишкин

Новосибирск 2020

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc54879005)

[ЗАДАНИЕ 3](#_Toc54879006)

[ОПИСАНИЕ РАБОТЫ 5](#_Toc54879007)

[ЗАКЛЮЧЕНИЕ 8](#_Toc54879008)

[Приложение 1 *Листинг файла Lab3.c* 9](#_Toc54879009)

[Приложение 2 *Листинг файла Lab3 (-m32 -O0).s* 10](#_Toc54879010)

[Приложение 3 *Листинг файла Lab3 (-m32 -O3).s* 11](#_Toc54879011)

[Приложение 4 *Листинг файла Lab3 (-m64 -O0).s* 12](#_Toc54879012)

[Приложение 5 *Листинг файла Lab3 (-m64 -O3).s* 13](#_Toc54879013)

# ЦЕЛЬ

* Знакомство с программной архитектурой x86/x86-64
* Анализ ассемблерного листинга программы для архитектуры x86/x86-64.

# ЗАДАНИЕ

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 (из лабораторной работы 1) сгенерировать ассемблерные листинги для архитектуры x86 и архитектуры x86-64, используя различные уровни комплексной оптимизации.
3. Проанализировать полученные листинги и сделать следующее.
   * Сопоставьте команды языка Си с машинными командами.
   * Определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти).
   * Описать и объяснить оптимизационные преобразования, выполненные компилятором.
   * Продемонстрировать использование ключевых особенностей архитектур x86 и x86-64 на конкретных участках ассемблерного кода.
   * Сравнить различия в программах для архитектуры x86 и архитектуры x86-64.
4. Составить отчет по лабораторной работе. Отчет должен содержать следующее.
   * Титульный лист.
   * Цель лабораторной работы.
   * Полный компилируемый листинг реализованной программы и команды для ее компиляции.
   * Листинг на ассемблере с описаниями назначения команд с точки зрения реализации алгоритма выбранного варианта.
   * Вывод по результатам лабораторной работы.

**Вариант задания: 7**

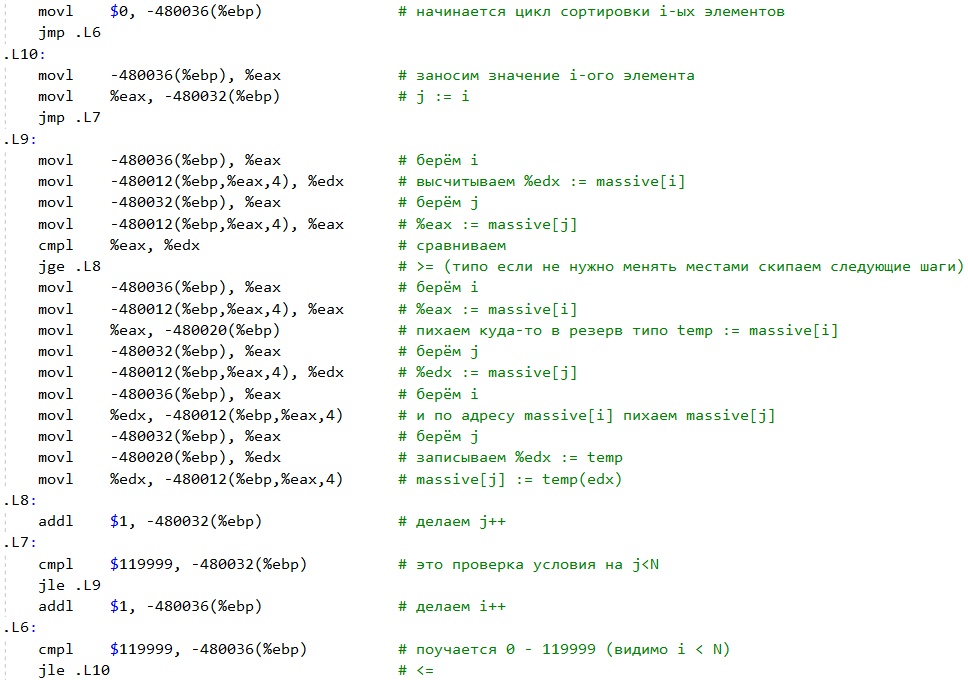
Алгоритм сортировки методом пузырька. Дан массив случайных чисел длины N. На первой итерации попарно упорядочиваются все соседние элементы; на второй – все элементы, кроме последнего элемента; на третьей – все элементы, кроме последнего элемента и предпоследнего элемента и т.п.

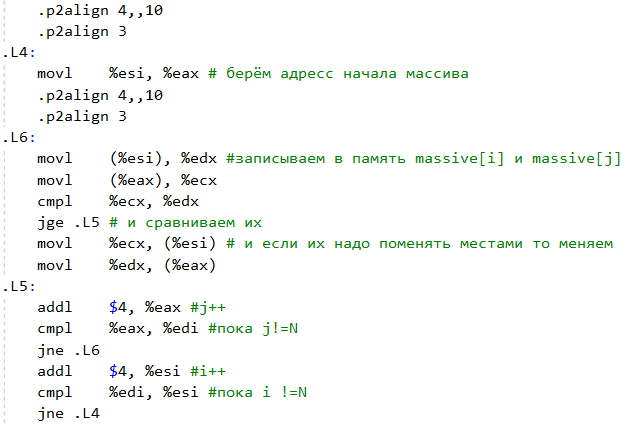
# **ОПИСАНИЕ РАБОТЫ**

За основу этой лабораторной работы была взята 1-ая лабораторная работа, написанная на языке Си (стандарта 1999г). Был написан алгоритм пузырьковой сортировки массива из N=120000 элементов. Входные данные этого массива генерируются рандомом (srand(1)).

Анализировал ассемблерный код четырёх видов оптимизации:

|  |  |  |  |
| --- | --- | --- | --- |
| Оптимизация | Команда для компиляции | Название файлов | Листинг |
| -m32 -O0 | gcc -S -m32 -O0 Lab3.c | Lab3 (-m32 -O0).s | [Приложение 2](#_Приложение_2) |
| -m32 -O3 | gcc -S -m32 -O3 Lab3.c | Lab3 (-m32 -O3).s | [Приложение 3](#_Приложение_3) |
| -m64 -O0 | gcc -S -m64 -O0 Lab3.c | Lab3 (-m64 -O0).s | [Приложение 4](#_Приложение_4) |
| -m64 -O3 | gcc -S -m64 -O3 Lab3.c | Lab3 (-m64 -O3).s | [Приложение 5](#_Приложение_5) |

Я подробно разобрал ассемблерский код ([Приложение 2](#_Приложение_2)). Поэтому не считаю нужным здесь его пересказывать. Могу отметить основные отличия между оптимизациями. Самое главное отличие – в логике сортировки пузырьком (на примере 32-битной версии):

* В первом случае инкрементация осуществляется за счёт счётчиков, благодаря которым потом высчитывается адрес ячейки, в то время как в оптимизированном случае сразу инкрементируется по адресам.
* Проверка условий осуществляется за счёт значения в счётчиках, а в оптимизированном случае – сравниваются адреса.
* И самое непонятное в неоптимизированном случае то, что при сравнении двух ячеек, несмотря на то что они уже записаны в регистрах, компилятор решил ещё раз их посчитать, тем самым явно делает одни и те же действия 2 раза. Также используется дополнительная переменная, как и в коде на языке Си. В оптимизированном случае осуществляется обмен значениями ячейки сразу: значения записываются в регистрах и при необходимости записываются, наоборот, в адреса, откуда были взяты (своеобразная замена, без использования дополнительных переменных).

В 64-битной версии используются дополнительный регистры, %r12 и %r13, а также некоторые другие, которые не существуют в 32-битной версии. Также интересен тот момент, что в начале программы и в конце её операнды имеют 8-байтовые длины.

И ещё интересный момент. Как вызывается функция «printf»:

|  |  |  |  |
| --- | --- | --- | --- |
| -m32 -O0 | -m32 -O3 | -m64 -O0 | -m64 -O3 |
|  |  |  |  |
|  |  |  |  |

# ЗАКЛЮЧЕНИЕ

В ходе этой лабораторной работы я лабораторной работы я познакомился с программной архитектурой x86/x86-64 и проанализировал ассемблерный листинг своей программы. Смог сопоставить команды языка си с машинным кодом. Определил размещение переменных языка Си в программах, указал основные оптимизации, и показал, чем отличаются эти оптимизации. Также я изучил набор регистров, а также их функции и назначение. Рассмотрел арифметико-логические команды, которые использовались в моей программе. Просмотрел способы адресации памяти и передачи управления. Разобрался в работе со стеком. Изучил как осуществляется вызов подпрограмм и их передачу параметров, а также способы возврата результатов. Немножко изучил работу сопроцессора и работу с векторными расширениями.

# Приложение 1

*Листинг файла Lab3.c*

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define N 120000

int main()

{

//Объявленные переменные

int massive[N]; //массив который будем сортировать

int start; //фиксирование времени начала сортировки

int finish; //фиксирование времени после сортировки

//Сохдаём массив с рандомными значениями в ячейках

srand(1);

for (int i = 0; i < N; i++)

{

massive[i] = rand();

}

printf("Before: ");

for (int i = 0; i < 5; i++)

{

printf("%d; ", massive[i]);

}

printf("...\n");

//Сортируем пузырьком

printf("Start sort...\n");

start = clock();

for (int i = 0; i < N; i++)

{

for (int j = i; j < N; j++)

{

if (massive[i] < massive[j])

{

int temp = massive[i];

massive[i] = massive[j];

massive[j] = temp;

}

}

}

finish = clock();

printf("Finish!\n");

//Выводим время

printf("Time = %d\n", (finish - start));

printf("After: ");

for (int i = 0; i < 5; i++)

{

printf("%d; ", massive[i]);

}

printf("...\n\n");

return 0;

}

# Приложение 2

*Листинг файла Lab3 (-m32 -O0).s*

.file "Lab3.c"

.section .rodata # секция данных только для чтения

.LC0:

.string "Before: "

.LC1:

.string "%d; "

.LC2:

.string "..."

.LC3:

.string "Start sort..."

.LC4:

.string "Finish!"

.LC5:

.string "Time = %d\n"

.LC6:

.string "After: "

.LC7:

.string "...\n"

.text # поместить следующее в сегмент кода (команды работают

только после .text)

.globl main # main - глобальный символ, видимый за пределами текущего

файла

.type main, @function # main - функция (а не данные)

main:

.LFB2:

.cfi\_startproc # задаёт начало процедуры и устанавливает начальный регистр

и смещение для расчёта адреса CFA (Canonical Frame Address)

leal 4(%esp), %ecx # поместить в %ecx адрес предыдущего элемента в стеке

.cfi\_def\_cfa 1, 0 # .cfi\_def\_cfa (регистр, смещение) определяет правило

вычисления CFA как: принимать адрес из регистра и добавить

к нему смещение

andl $-16, %esp # конъюкция числа -16 (11110000) \*

pushl -4(%ecx) # поместили на вершину стека значение элемента в стеке

4\*ecx

pushl %ebp # запоминаем текущее значение регистра %ebp (потом будем ссылаться на него)

.cfi\_escape 0x10,0x5,0x2,0x75,0 # позволяет пользователю добавлять произвольные байты к

информации раскрутки

movl %esp, %ebp # записываем текущее положение вершины стека

pushl %ecx # кладём наверх стека значение %ecx

.cfi\_escape 0xf,0x3,0x75,0x7c,0x6 # позволяет пользователю добавлять произвольные байты к информации раскрутки

subl $480052, %esp # резервируем место для локальных переменных вверх (скорее всего для массива)

movl %gs:20, %eax # заносим в %eax значение сегментного регистра (дополнительный сегментный регистр без специального назначения; в этих регистрах содержатся адреса ячеек памяти, с которых начинаются соответствующие сегменты )

movl %eax, -12(%ebp) # заносим %gs в стек

xorl %eax, %eax # обнуляем

subl $12, %esp # поднимаемся выше

pushl $1 # заносим единичку

call srand # вызываем системную функцию рандома с сидом 1

addl $16, %esp # возращаемся вниз

movl $0, -480044(%ebp) # начинаем цикл заполением рандомом (i=0)

jmp .L2

.L3: # а тут заполнение цикла

call rand

movl %eax, %edx # eax(rand)=:edx

movl -480044(%ebp), %eax # eax=i

movl %edx, -480012(%ebp,%eax,4) # (%ebp + %eax\*4 -480012) := %edx massive[i] = rand()

addl $1, -480044(%ebp) # i++ наверное сдвигаем ячейку на следующую

.L2:

cmpl $119999, -480044(%ebp) # поучается 0(i) - 119999

jle .L3 # <= (проверка условия выполнения цикла)

subl $12, %esp # бежим по стеку

pushl $.LC0 # пихаем строчку

call printf # выводим её

addl $16, %esp # вощращаемся в стек обратно

movl $0, -480040(%ebp) # начинается цикл вывода первых 5и элементов (i=0)

jmp .L4

.L5:

movl -480040(%ebp), %eax # сейвим i-ое

movl -480012(%ebp,%eax,4), %eax # (%ebp + %eax\*4 -480012) =: %eax присваеваем i-ое значение

subl $8, %esp # поднимаемся на 8 вверх

pushl %eax # пихаем i-ый элемент на вывод

pushl $.LC1 # берём строчку для вывода

call printf # выводим её

addl $16, %esp # спускаемся обратно

addl $1, -480040(%ebp) # смещаемся (делаем i+1)

.L4:

cmpl $4, -480040(%ebp) # 0 - 4

jle .L5 # <=

subl $12, %esp # поднимаемся вверх

pushl $.LC2 # берём строчку

call puts # обозначает вывод строки + символ "\n"

addl $16, %esp # спускаемся на 16

subl $12, %esp # поднимаемся на 12

pushl $.LC3 # берём строчку

call puts # обозначает вывод строки + символ "\n"

addl $16, %esp # спускаемся на 16

call clock # вызываем системное время

movl %eax, -480024(%ebp) # в стеке фиксируем начало времени start = clock()

movl $0, -480036(%ebp) # начинается цикл сортировки i-ых элементов

jmp .L6

.L10:

movl -480036(%ebp), %eax # заносим значение i-ого элемента

movl %eax, -480032(%ebp) # j := i

jmp .L7

.L9:

movl -480036(%ebp), %eax # берём i

movl -480012(%ebp,%eax,4), %edx # высчитываем %edx := massive[i]

movl -480032(%ebp), %eax # берём j

movl -480012(%ebp,%eax,4), %eax # %eax := massive[j]

cmpl %eax, %edx # сравниваем

jge .L8 # >= (типо если не нужно менять местами скипаем следующие

шаги)

movl -480036(%ebp), %eax # берём i

movl -480012(%ebp,%eax,4), %eax # %eax := massive[i]

movl %eax, -480020(%ebp) # пихаем куда-то в резерв типо temp := massive[i]

movl -480032(%ebp), %eax # берём j

movl -480012(%ebp,%eax,4), %edx # %edx := massive[j]

movl -480036(%ebp), %eax # берём i

movl %edx, -480012(%ebp,%eax,4) # и по адресу massive[i] пихаем massive[j]

movl -480032(%ebp), %eax # берём j

movl -480020(%ebp), %edx # записываем %edx := temp

movl %edx, -480012(%ebp,%eax,4) # massive[j] := temp(edx)

.L8:

addl $1, -480032(%ebp) # делаем j++

.L7:

cmpl $119999, -480032(%ebp) # это проверка условия на j<N

jle .L9

addl $1, -480036(%ebp) # делаем i++

.L6:

cmpl $119999, -480036(%ebp) # поучается 0 - 119999 (видимо i < N)

jle .L10 # <=

call clock # вызваем ситемное время

movl %eax, -480016(%ebp) # фиксируем время finish = clock()

subl $12, %esp # по стеку вверх

pushl $.LC4 # выводми "Finish!"

call puts # делаем "\n"

addl $16, %esp # возращаемся обратно

movl -480016(%ebp), %eax # берём время finish =: %eax

subl -480024(%ebp), %eax # finish - start

subl $8, %esp # поднимаем стек

pushl %eax # записываем результат

pushl $.LC5 # готовим строчку

call printf # вызываем вывод

addl $16, %esp # спускаемся на 16

subl $12, %esp # поднимаемся на 12

pushl $.LC6 # выводим After:

call printf # вызываем вывод

addl $16, %esp # поднимаемся снова обратно

movl $0, -480028(%ebp) # начинаем цикл вывода 5и отсортированных элементов (i=0)

jmp .L11

.L12:

movl -480028(%ebp), %eax # %eax := i

movl -480012(%ebp,%eax,4), %eax # %eax := massive[i]

subl $8, %esp # поднимаемся чуть выше

pushl %eax # пихаем massive[i] видимо для вывода

pushl $.LC1 # пушим строчку

call printf # вызываем системный вывод

addl $16, %esp # спускаемся на 16

addl $1, -480028(%ebp) # делаем i++

.L11:

cmpl $4, -480028(%ebp) # 0 - 4

jle .L12 # <=

subl $12, %esp # перемещаемся чуть выше

pushl $.LC7 # вызываем строчку

call puts # пихаем строчку + "\n"

addl $16, %esp # возращаемся вниз

movl $0, %eax # %eax = 0

movl -12(%ebp), %ecx # вспомниаем про сегментный регистр %ecx = %gs:20

xorl %gs:20, %ecx # скорее всего проверяем уго равнество на обнулении

je .L14 # и если они эквиваленты

call \_\_stack\_chk\_fail # чекает переполнение стека ("Не проверяет переполнение

стека. Он просто сообщает один при использовании")

.L14:

movl -4(%ebp), %ecx # что-то системное записывает

.cfi\_def\_cfa 1, 0 # .cfi\_def\_cfa (регистр, смещение) определяет правило

вычисления CFA как: принимать адрес из регистра и добавить

к нему смещение

leave # (movl %ebp %esp; pop %ebp)

.cfi\_restore 5 # .cfi\_restore говорит, что правило для регистра теперь

такое же, как и в начале функции, после того, как были выполнены все начальные инструкции, добавленные .cfi\_startproc

leal -4(%ecx), %esp # орять вернули что-то на стек

.cfi\_def\_cfa 4, 4 # .cfi\_def\_cfa (регистр, смещение) определяет правило

вычисления CFA как: принимать адрес из регистра и добавить к нему смещение

ret # (pop %eax; jmp %eax) программа закончилась (return 0; в

main)

.cfi\_endproc # закрывает свою запись очистки, ранее открытую

.cfi\_startproc, и передает ее в .eh\_frame.

.LFE2:

.size main, .-main

.ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"

.section .note.GNU-stack,"",@progbits

# --------------------------------------------------------------------------------------------------

# .eh\_frame - предоставление дополнительной информации в среду выполнения для описания фреймов вызовов, которые должны быть размотаны во время обработки исключения

# (\*) При передаче управления процедуре микропроцессор автоматически записывает в вершину стека 4 байта. Эти байты являются адресом возврата в вызывающую программу. Если перед передачей управления процедуре командой call в стек были записаны переданные процедуре данные или указатели на них, то они окажутся под адресом возврата

# Приложение 3

*Листинг файла Lab3 (-m32 -O3).s*

.file "Lab3.c"

.section .rodata.str1.1,"aMS",@progbits,1

.LC0:

.string "Before: "

.LC1:

.string "%d; "

.LC2:

.string "..."

.LC3:

.string "Start sort..."

.LC4:

.string "Finish!"

.LC5:

.string "Time = %d\n"

.LC6:

.string "After: "

.LC7:

.string "...\n"

.section .text.unlikely,"ax",@progbits

.LCOLDB8:

.section .text.startup,"ax",@progbits

.LHOTB8:

.p2align 4,,15

.globl main

.type main, @function

main:

.LFB38:

.cfi\_startproc

leal 4(%esp), %ecx

.cfi\_def\_cfa 1, 0

andl $-16, %esp

pushl -4(%ecx)

pushl %ebp

.cfi\_escape 0x10,0x5,0x2,0x75,0

movl %esp, %ebp

pushl %edi

pushl %esi

pushl %ebx

pushl %ecx

.cfi\_escape 0xf,0x3,0x75,0x70,0x6

.cfi\_escape 0x10,0x7,0x2,0x75,0x7c

.cfi\_escape 0x10,0x6,0x2,0x75,0x78

.cfi\_escape 0x10,0x3,0x2,0x75,0x74

leal -480028(%ebp), %esi #сохраняем адреса границ массива

leal -28(%ebp), %edi #от edi до esi вверх по стеку

subl $480036, %esp

movl %esi, %ebx

pushl $1

movl %gs:20, %eax

movl %eax, -28(%ebp)

xorl %eax, %eax

call srand #начинается программа

addl $16, %esp

.p2align 4,,10 #https://sourceware.org/binutils/docs/as/P2align.html#P2align

.p2align 3

.L2:

call rand

addl $4, %ebx #i+1 (по адресу)

movl %eax, -4(%ebx) #зваписываем massive[i-1]

cmpl %ebx, %edi #сравниваем пока не не дойдём доконца

jne .L2

subl $8, %esp

xorl %ebx, %ebx

pushl $.LC0

pushl $1

call \_\_printf\_chk

addl $16, %esp

.L3:

subl $4, %esp #будем выводить первые 5 элементов

pushl -480028(%ebp,%ebx,4) #massive[i]

addl $1, %ebx #i++

pushl $.LC1

pushl $1

call \_\_printf\_chk

addl $16, %esp

cmpl $5, %ebx #i<5

jne .L3 #i!=5

subl $12, %esp

pushl $.LC2 #"..."

call puts

movl $.LC3, (%esp) # "Start sort"

call puts

call clock

addl $16, %esp

movl %eax, %ebx #фиксируем время страрта

.p2align 4,,10

.p2align 3

.L4:

movl %esi, %eax # берём адресс начала массива

.p2align 4,,10

.p2align 3

.L6:

movl (%esi), %edx #записываем в память massive[i] и massive[j]

movl (%eax), %ecx

cmpl %ecx, %edx

jge .L5 # и сравниваем их

movl %ecx, (%esi) # и если их надо поменять местами то меняем

movl %edx, (%eax)

.L5:

addl $4, %eax #j++

cmpl %eax, %edi #пока j!=N

jne .L6

addl $4, %esi #i++

cmpl %edi, %esi #пока i !=N

jne .L4

call clock #фиксируем финишное время

subl $12, %esp

movl %eax, %esi #finish

pushl $.LC4

subl %ebx, %esi #finish - start

xorl %ebx, %ebx

call puts

addl $12, %esp

pushl %esi

pushl $.LC5

pushl $1

call \_\_printf\_chk #выводим время

popl %eax

popl %edx

pushl $.LC6 #"After"

pushl $1

call \_\_printf\_chk

addl $16, %esp

.L8:

subl $4, %esp #начинаем выводить 5 элементов

pushl -480028(%ebp,%ebx,4) #massive[i]

addl $1, %ebx #i++

pushl $.LC1

pushl $1

call \_\_printf\_chk

addl $16, %esp

addl $16, %esp

cmpl $5, %ebx #i!=5

jne .L8

subl $12, %esp #просто выводим первод новой строки

pushl $.LC7

call puts

addl $16, %esp

xorl %eax, %eax #тут идёт массовая зачитска перед выходом программы

movl -28(%ebp), %ebx

xorl %gs:20, %ebx

jne .L16

leal -16(%ebp), %esp

popl %ecx

.cfi\_remember\_state

.cfi\_restore 1

.cfi\_def\_cfa 1, 0

popl %ebx

.cfi\_restore 3

popl %esi

.cfi\_restore 6

popl %edi

.cfi\_restore 7

popl %ebp

.cfi\_restore 5

leal -4(%ecx), %esp

.cfi\_def\_cfa 4, 4

ret

.L16:

.cfi\_restore\_state

call \_\_stack\_chk\_fail

.cfi\_endproc

.LFE38:

.size main, .-main

.section .text.unlikely

.LCOLDE8:

.section .text.startup

.LHOTE8:

.ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"

.section .note.GNU-stack,"",@progbits

# Приложение 4

*Листинг файла Lab3 (-m64 -O0).s*

.file "Lab3.c"

.section .rodata

.LC0:

.string "Before: "

.LC1:

.string "%d; "

.LC2:

.string "..."

.LC3:

.string "Start sort..."

.LC4:

.string "Finish!"

.LC5:

.string "Time = %d\n"

.LC6:

.string "After: "

.LC7:

.string "...\n"

.text

.globl main

.type main, @function

main:

.LFB2:

.cfi\_startproc

pushq %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.cfi\_def\_cfa\_register 6

subq $480048, %rsp #резервируем память для массива и прочих элемментов

movq %fs:40, %rax

movq %rax, -8(%rbp)

xorl %eax, %eax

movl $1, %edi #srand(1)

call srand

movl $0, -480048(%rbp)

jmp .L2

.L3: #заполняем цикл рандомами

call rand

movl %eax, %edx

movl -480048(%rbp), %eax

cltq #https://sourceware.org/binutils/docs/as/i386\_002dMnemonics.html (sign-extend dword in ‘%eax’ to quad in ‘%rax’)

movl %edx, -480016(%rbp,%rax,4)

addl $1, -480048(%rbp)

.L2:

cmpl $119999, -480048(%rbp)

jle .L3

movl $.LC0, %edi

movl $0, %eax

call printf

movl $0, -480044(%rbp) #вывод первых 5-и элементов

jmp .L4

.L5:

movl -480044(%rbp), %eax

cltq

movl -480016(%rbp,%rax,4), %eax

movl %eax, %esi

movl $.LC1, %edi

movl $0, %eax

call printf

addl $1, -480044(%rbp)

.L4:

cmpl $4, -480044(%rbp)

jle .L5

movl $.LC2, %edi

call puts

movl $.LC3, %edi

call puts

call clock

movl %eax, -480028(%rbp) #фиксируем время start

movl $0, -480040(%rbp) #начало сортировки

jmp .L6

.L10:

movl -480040(%rbp), %eax

movl %eax, -480036(%rbp) # j := i

jmp .L7

.L9:

movl -480040(%rbp), %eax #i

cltq

movl -480016(%rbp,%rax,4), %edx #massive[i]

movl -480036(%rbp), %eax #j

cltq

movl -480016(%rbp,%rax,4), %eax #massive[j]

cmpl %eax, %edx

jge .L8 # >= то меняем

movl -480040(%rbp), %eax

cltq

movl -480016(%rbp,%rax,4), %eax

movl %eax, -480024(%rbp)

movl -480036(%rbp), %eax

cltq

movl -480016(%rbp,%rax,4), %edx

movl -480040(%rbp), %eax

cltq

movl %edx, -480016(%rbp,%rax,4)

movl -480036(%rbp), %eax

cltq

movl -480024(%rbp), %edx

movl %edx, -480016(%rbp,%rax,4)

.L8:

addl $1, -480036(%rbp) # j++

.L7:

cmpl $119999, -480036(%rbp) # j<N

jle .L9

addl $1, -480040(%rbp) #i++

.L6:

cmpl $119999, -480040(%rbp) # i<N

jle .L10

call clock

movl %eax, -480020(%rbp) # фиксируем finish

movl $.LC4, %edi

call puts

movl -480020(%rbp), %eax

subl -480028(%rbp), %eax # finish - start

movl %eax, %esi

movl $.LC5, %edi

movl $0, %eax

call printf

movl $.LC6, %edi

movl $0, %eax

call printf

movl $0, -480032(%rbp) #выводим первые 5 элементов

jmp .L11

.L12:

movl -480032(%rbp), %eax

cltq

movl -480016(%rbp,%rax,4), %eax

movl %eax, %esi

movl $.LC1, %edi

movl $0, %eax

call printf

addl $1, -480032(%rbp)

.L11:

cmpl $4, -480032(%rbp)

jle .L12

movl $.LC7, %edi

call puts

movl $0, %eax

movq -8(%rbp), %rcx

xorq %fs:40, %rcx

je .L14

call \_\_stack\_chk\_fail

.L14:

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE2:

.size main, .-main

.ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"

.section .note.GNU-stack,"",@progbits

# Приложение 5

*Листинг файла Lab3 (-m64 -O3).s*

.file "Lab3.c"

.section .rodata.str1.1,"aMS",@progbits,1

.LC0:

.string "Before: "

.LC1:

.string "%d; "

.LC2:

.string "..."

.LC3:

.string "Start sort..."

.LC4:

.string "Finish!"

.LC5:

.string "Time = %d\n"

.LC6:

.string "After: "

.LC7:

.string "...\n"

.section .text.unlikely,"ax",@progbits

.LCOLDB8:

.section .text.startup,"ax",@progbits

.LHOTB8:

.p2align 4,,15

.globl main

.type main, @function

main:

.LFB38:

.cfi\_startproc

pushq %r13

.cfi\_def\_cfa\_offset 16

.cfi\_offset 13, -16

pushq %r12

.cfi\_def\_cfa\_offset 24

.cfi\_offset 12, -24

movl $1, %edi

pushq %rbp

.cfi\_def\_cfa\_offset 32

.cfi\_offset 6, -32

pushq %rbx

.cfi\_def\_cfa\_offset 40

.cfi\_offset 3, -40

subq $480024, %rsp

.cfi\_def\_cfa\_offset 480064

leaq 480000(%rsp), %r12 #запомнием адресок конца массива

movq %rsp, %rbx #начала масстива

movq %rsp, %rbp

movq %fs:40, %rax

movq %rax, 480008(%rsp)

xorl %eax, %eax

call srand #начинается программа

.p2align 4,,10

.p2align 3

.L2:

call rand

addq $4, %rbp

movl %eax, -4(%rbp)

cmpq %rbp, %r12

jne .L2

leaq 20(%rsp), %rbp

movl $.LC0, %esi

movl $1, %edi

xorl %eax, %eax

movq %rsp, %r12

call \_\_printf\_chk

.L3:

movl (%r12), %edx #будем выводить первые 5 элементов

xorl %eax, %eax

movl $.LC1, %esi

movl $1, %edi

addq $4, %r12

call \_\_printf\_chk

cmpq %rbp, %r12

jne .L3

movl $.LC2, %edi

call puts

movl $.LC3, %edi

call puts

call clock

xorl %edi, %edi

movq %rax, %r12 #фиксируем время страрта

.p2align 4,,10

.p2align 3

.L4: #начинаем сортировать

movl %edi, %eax

.p2align 4,,10

.p2align 3

.L6:

movslq %eax, %rdx

movl (%rsp,%rdi,4), %ecx #записываем в память massive[i] и massive[j]

movl (%rsp,%rdx,4), %esi

cmpl %esi, %ecx

jge .L5 # и сравниваем их

movl %esi, (%rsp,%rdi,4)

movl %ecx, (%rsp,%rdx,4)

.L5:

addl $1, %eax #j++

cmpl $120000, %eax

jne .L6

addq $1, %rdi #i++

cmpq $120000, %rdi

jne .L4

call clock #фиксируем финишное время

movl $.LC4, %edi

movq %rax, %r13 #finish

call puts

movl %r13d, %edx #finish - start

movl $.LC5, %esi

movl $1, %edi

subl %r12d, %edx

xorl %eax, %eax

call \_\_printf\_chk #выводим время

movl $.LC6, %esi

movl $1, %edi

xorl %eax, %eax

call \_\_printf\_chk

.L8:

movl (%rbx), %edx #начинаем выводить 5 элементов

xorl %eax, %eax

movl $.LC1, %esi

movl $1, %edi

addq $4, %rbx

call \_\_printf\_chk

cmpq %rbx, %rbp

jne .L8

movl $.LC7, %edi

call puts

xorl %eax, %eax

movq 480008(%rsp), %rbx

xorq %fs:40, %rbx

jne .L16

addq $480024, %rsp

.cfi\_remember\_state

.cfi\_def\_cfa\_offset 40

popq %rbx

.cfi\_def\_cfa\_offset 32

popq %rbp

.cfi\_def\_cfa\_offset 24

popq %r12

.cfi\_def\_cfa\_offset 16

popq %r13

.cfi\_def\_cfa\_offset 8

ret

.L16:

.cfi\_restore\_state

call \_\_stack\_chk\_fail

.cfi\_endproc

.LFE38:

.size main, .-main

.section .text.unlikely

.LCOLDE8:

.section .text.startup

.LHOTE8:

.ident "GCC: (Ubuntu 5.5.0-12ubuntu1~16.04) 5.5.0 20171010"

.section .note.GNU-stack,"",@progbits