МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

## «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №8

по дисциплине: Архитектура вычислительных систем

тема: «Способы вызова ассемблерных подпрограмм

в языках высокого уровня»

# Выполнил: ст. группы ПВ-223

# Дмитриев Андрей Александрович

Проверил:

# Осипов Олег Васильевич

Белгород 2024 г.

# Вариант 2

Цель работы: изучение способов вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек.

|  |
| --- |
| Варианты 1-7  Сортировка части массива чисел с индексами от *start* до *end* включительно. Отсортированный массив (элементы *start*…*end*) записать в *res* и возвратить его адрес. Исходный массив *a* оставить без изменений. Под массив *res* зарезервировать память в необходимом размере, но не больше, чем нужно.  Пример: *a* = {4, 5, 4, 2, **5, 7, 5, 6, 5**, 3, 5, 6}, *start* = 4, *end* = 8;  *res* = {5, 5, 5, 6, 7} (сортировка по не убыванию). Длина массива *res* равна 5. |
| Пузырьковая сортировка по не возрастанию.  double\* sort(double\* res, double\* a, int start, int end). |

Ход работы:

Код программ:

Функции на ассемблере:

.686

.model flat, stdcall

option casemap: none

; include c:\masm32\include\windows.inc

; include c:\masm32\include\kernel32.inc

; include c:\masm32\include\user32.inc

include c:\masm32\include\msvcrt.inc

include c:\masm32\include\masm32rt.inc

includelib c:\masm32\lib\kernel32.lib

includelib c:\masm32\lib\msvcrt.lib

includelib c:\masm32\lib\user32.lib

.code

DllMain proc hlnstDLL:dword, reason: dword, unused: dword

mov eax, 1

ret

DllMain endp

; double\* sort(double\* res, double\* a, int start, int end).

sort\_stdcall proc

    ; Сохранение регистров

    push ebp

    mov ebp, esp

    add ebp, 8

    push ebx

    push esi

    push edi

    push ecx

    push edx

    ; Извлечение аргументов из стека

    mov edi, [ebp]  ; Указатель на результирующий массив res

    mov esi, [ebp + 4] ; Указатель на исходный массив a

    mov ebx, [ebp + 8] ; Индекс начала start

    mov ecx, [ebp + 12] ; Индекс конца end

    ; Копирование подмассива в res

    sub ecx, ebx        ; Размер подмассива

    inc ecx             ; Увеличиваем на 1, чтобы включить end

    mov edx, 0          ; Индекс в res

copy\_loop:

    push ecx

    mov eax, [esi + ebx \* 8] ; Загрузка элемента из a[start]

    mov ecx, [esi + ebx \* 8 + 4]

    mov [edi + edx \* 8], eax ; Сохранение в res

    mov [edi + edx \* 8 + 4], ecx

    pop ecx

    inc ebx                  ; Следующий индекс в a

    inc edx                  ; Следующий индекс в res

    loop copy\_loop           ; Повторяем для всех элементов

    finit

    ; Сортировка пузырьком

    mov ecx, [ebp + 12]

    sub ecx, [ebp + 8]      ; Размер подмассива

    inc ecx                  ; Увеличиваем на 1

sort\_outer\_loop:

    push ecx                 ; Сохраняем счетчик внешнего цикла

    mov edx, 0               ; Индекс для внутреннего цикла

sort\_inner\_loop:

    fld qword ptr [edi + edx \* 8] ; Загрузка числа с плавающей точкой в стек FPU

    fld qword ptr [edi + edx \* 8 + 8] ; Загрузка следующего числа с плавающей точкой в стек FPU

    fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

    fstsw ax ; Сохранение слова состояния FPU в AX

    sahf ; Загрузка AH в регистр флагов

    jbe no\_swap ; Переход, если текущий элемент <= следующего

    fld qword ptr [edi + edx \* 8] ; Обмен значениями

    fld qword ptr [edi + edx \* 8 + 8]

    fstp qword ptr [edi + edx \* 8]

    fstp qword ptr [edi + edx \* 8 + 8]

no\_swap:

    inc edx                  ; Следующий индекс

    loop sort\_inner\_loop     ; Повторяем для всех элементов

    pop ecx                  ; Восстанавливаем счетчик внешнего цикла

    loop sort\_outer\_loop     ; Повторяем для всех элементов

    ; Восстановление регистров

    pop edx

    pop ecx

    pop edi

    pop esi

    pop ebx

    ; Возврат указателя на res

    mov eax, [ebp]

    pop ebp

    ret 16                  ; Очистка стека (4 аргумента \* 4 байта)

sort\_stdcall endp

; double\* sort(double\* res, double\* a, int start, int end).

sort\_cdecl proc

    ; Сохранение регистров

    push ebp

    mov ebp, esp

    add ebp, 8

    push ebx

    push esi

    push edi

    push ecx

    push edx

    ; Извлечение аргументов из стека

    mov edi, [ebp]  ; Указатель на результирующий массив res

    mov esi, [ebp + 4] ; Указатель на исходный массив a

    mov ebx, [ebp + 8] ; Индекс начала start

    mov ecx, [ebp + 12] ; Индекс конца end

    ; Копирование подмассива в res

    sub ecx, ebx        ; Размер подмассива

    inc ecx             ; Увеличиваем на 1, чтобы включить end

    mov edx, 0          ; Индекс в res

copy\_loop:

    push ecx

    mov eax, [esi + ebx \* 8] ; Загрузка элемента из a[start]

    mov ecx, [esi + ebx \* 8 + 4]

    mov [edi + edx \* 8], eax ; Сохранение в res

    mov [edi + edx \* 8 + 4], ecx

    pop ecx

    inc ebx                  ; Следующий индекс в a

    inc edx                  ; Следующий индекс в res

    loop copy\_loop           ; Повторяем для всех элементов

    finit

    ; Сортировка пузырьком

    mov ecx, [ebp + 12]

    sub ecx, [ebp + 8]      ; Размер подмассива

    inc ecx                  ; Увеличиваем на 1

sort\_outer\_loop:

    push ecx                 ; Сохраняем счетчик внешнего цикла

    mov edx, 0               ; Индекс для внутреннего цикла

sort\_inner\_loop:

    fld qword ptr [edi + edx \* 8] ; Загрузка числа с плавающей точкой в стек FPU

    fld qword ptr [edi + edx \* 8 + 8] ; Загрузка следующего числа с плавающей точкой в стек FPU

    fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

    fstsw ax ; Сохранение слова состояния FPU в AX

    sahf ; Загрузка AH в регистр флагов

    jbe no\_swap ; Переход, если текущий элемент <= следующего

    fld qword ptr [edi + edx \* 8] ; Обмен значениями

    fld qword ptr [edi + edx \* 8 + 8]

    fstp qword ptr [edi + edx \* 8]

    fstp qword ptr [edi + edx \* 8 + 8]

no\_swap:

    inc edx                  ; Следующий индекс

    loop sort\_inner\_loop     ; Повторяем для всех элементов

    pop ecx                  ; Восстанавливаем счетчик внешнего цикла

    loop sort\_outer\_loop     ; Повторяем для всех элементов

    ; Восстановление регистров

    pop edx

    pop ecx

    pop edi

    pop esi

    pop ebx

    ; Возврат указателя на res

    mov eax, [ebp]

    pop ebp

    ret                   ; Стек не очищаем

sort\_cdecl endp

; double\* sort(double\* res, double\* a, int start, int end).

sort\_fastcall proc

    ; Сохранение регистров

    push ebp

    mov ebp, esp

    add ebp, 8

    push ebx

    push esi

    push edi

    push ecx

    push edx

    ; Извлечение аргументов из регистров и стека

    mov edi, ecx        ; Указатель на результирующий массив res

    mov esi, edx        ; Указатель на исходный массив а

    mov ebx, [ebp]  ; Индекс начала start

    mov ecx, [ebp + 4] ; Индекс конца end

    ; Копирование подмассива в res

    sub ecx, ebx        ; Размер подмассива

    inc ecx             ; Увеличиваем на 1, чтобы включить end

    mov edx, 0          ; Индекс в res

copy\_loop:

    push ecx

    mov eax, [esi + ebx \* 8] ; Загрузка элемента из a[start]

    mov ecx, [esi + ebx \* 8 + 4]

    mov [edi + edx \* 8], eax ; Сохранение в res

    mov [edi + edx \* 8 + 4], ecx

    pop ecx

    inc ebx                  ; Следующий индекс в a

    inc edx                  ; Следующий индекс в res

    loop copy\_loop           ; Повторяем для всех элементов

    ; Сортировка пузырьком

    mov ecx, [ebp + 4]

    sub ecx, [ebp]           ; Размер подмассива

    inc ecx                  ; Увеличиваем на 1

    finit

sort\_outer\_loop:

    push ecx                 ; Сохраняем счетчик внешнего цикла

    mov edx, 0               ; Индекс для внутреннего цикла

sort\_inner\_loop:

    fld qword ptr [edi + edx \* 8] ; Загрузка числа с плавающей точкой в стек FPU

    fld qword ptr [edi + edx \* 8 + 8] ; Загрузка следующего числа с плавающей точкой в стек FPU

    fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

    fstsw ax ; Сохранение слова состояния FPU в AX

    sahf ; Загрузка AH в регистр флагов

    jbe no\_swap ; Переход, если текущий элемент <= следующего

    fld qword ptr [edi + edx \* 8] ; Обмен значениями

    fld qword ptr [edi + edx \* 8 + 8]

    fstp qword ptr [edi + edx \* 8]

    fstp qword ptr [edi + edx \* 8 + 8]

no\_swap:

    inc edx                  ; Следующий индекс

    loop sort\_inner\_loop     ; Повторяем для всех элементов

    pop ecx                  ; Восстанавливаем счетчик внешнего цикла

    loop sort\_outer\_loop     ; Повторяем для всех элементов

    ; Восстановление регистров

    pop edx

    pop ecx

    pop edi

    pop esi

    pop ebx

    ; Возврат указателя на res

    mov eax, ecx

    pop ebp

    ret 8                  ; Очистка стека (2 аргумента \* 4 байта)

sort\_fastcall endp

; double\* sort(double\* res, double\* a, int start, int end).

sort\_cdecl\_args proc c \_res:DWORD, \_a:DWORD, \_start\_:DWORD, \_end\_:DWORD

    ; Сохранение регистров

    push ebx

    push esi

    push edi

    push ecx

    push edx

    ; Извлечение аргументов из стека

    mov edi, \_res        ; Указатель на результирующий массив res

    mov esi, \_a          ; Указатель на исходный массив a

    mov ebx, \_start\_     ; Индекс начала start

    mov ecx, \_end\_       ; Индекс конца end

    ; Копирование подмассива в res

    sub ecx, ebx        ; Размер подмассива

    inc ecx             ; Увеличиваем на 1, чтобы включить end

    mov edx, 0          ; Индекс в res

copy\_loop:

    push ecx

    mov eax, [esi + ebx \* 8] ; Загрузка элемента из a[start]

    mov ecx, [esi + ebx \* 8 + 4]

    mov [edi + edx \* 8], eax ; Сохранение в res

    mov [edi + edx \* 8 + 4], ecx

    pop ecx

    inc ebx                  ; Следующий индекс в a

    inc edx                  ; Следующий индекс в res

    loop copy\_loop           ; Повторяем для всех элементов

    finit

    ; Сортировка пузырьком

    mov ecx, \_end\_

    sub ecx, \_start\_          ; Размер подмассива

    inc ecx                  ; Увеличиваем на 1

sort\_outer\_loop:

    push ecx                 ; Сохраняем счетчик внешнего цикла

    mov edx, 0               ; Индекс для внутреннего цикла

sort\_inner\_loop:

    fld qword ptr [edi + edx \* 8] ; Загрузка числа с плавающей точкой в стек FPU

    fld qword ptr [edi + edx \* 8 + 8] ; Загрузка следующего числа с плавающей точкой в стек FPU

    fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

    fstsw ax ; Сохранение слова состояния FPU в AX

    sahf ; Загрузка AH в регистр флагов

    jbe no\_swap ; Переход, если текущий элемент <= следующего

    fld qword ptr [edi + edx \* 8] ; Обмен значениями

    fld qword ptr [edi + edx \* 8 + 8]

    fstp qword ptr [edi + edx \* 8]

    fstp qword ptr [edi + edx \* 8 + 8]

no\_swap:

    inc edx                  ; Следующий индекс

    loop sort\_inner\_loop     ; Повторяем для всех элементов

    pop ecx                  ; Восстанавливаем счетчик внешнего цикла

    loop sort\_outer\_loop     ; Повторяем для всех элементов

    ; Восстанавливаем регистры

    pop edx

    pop ecx

    pop edi

    pop esi

    pop ebx

    ; Возврат указателя на res

    mov eax, \_res

    ret

sort\_cdecl\_args endp

; double\* sort(double\* res, double\* a, int start, int end).

sort\_stdcall\_args proc stdcall \_res:DWORD, \_a:DWORD, \_start\_:DWORD, \_end\_:DWORD

    ; Сохранение регистров

    push ebx

    push esi

    push edi

    push ecx

    push edx

    ; Извлечение аргументов из стека

    mov edi, \_res        ; Указатель на результирующий массив res

    mov esi, \_a          ; Указатель на исходный массив a

    mov ebx, \_start\_     ; Индекс начала start

    mov ecx, \_end\_       ; Индекс конца end

    ; Копирование подмассива в res

    sub ecx, ebx        ; Размер подмассива

    inc ecx             ; Увеличиваем на 1, чтобы включить end

    mov edx, 0          ; Индекс в res

copy\_loop:

    push ecx

    mov eax, [esi + ebx \* 8] ; Загрузка элемента из a[start]

    mov ecx, [esi + ebx \* 8 + 4]

    mov [edi + edx \* 8], eax ; Сохранение в res

    mov [edi + edx \* 8 + 4], ecx

    pop ecx

    inc ebx                  ; Следующий индекс в a

    inc edx                  ; Следующий индекс в res

    loop copy\_loop           ; Повторяем для всех элементов

    finit

    ; Сортировка пузырьком

    mov ecx, \_end\_

    sub ecx, \_start\_          ; Размер подмассива

    inc ecx                  ; Увеличиваем на 1

sort\_outer\_loop:

    push ecx                 ; Сохраняем счетчик внешнего цикла

    mov edx, 0               ; Индекс для внутреннего цикла

sort\_inner\_loop:

    fld qword ptr [edi + edx \* 8] ; Загрузка числа с плавающей точкой в стек FPU

    fld qword ptr [edi + edx \* 8 + 8] ; Загрузка следующего числа с плавающей точкой в стек FPU

    fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

    fstsw ax ; Сохранение слова состояния FPU в AX

    sahf ; Загрузка AH в регистр флагов

    jbe no\_swap ; Переход, если текущий элемент <= следующего

    fld qword ptr [edi + edx \* 8] ; Обмен значениями

    fld qword ptr [edi + edx \* 8 + 8]

    fstp qword ptr [edi + edx \* 8]

    fstp qword ptr [edi + edx \* 8 + 8]

no\_swap:

    inc edx                  ; Следующий индекс

    loop sort\_inner\_loop     ; Повторяем для всех элементов

    pop ecx                  ; Восстанавливаем счетчик внешнего цикла

    loop sort\_outer\_loop     ; Повторяем для всех элементов

    ; Восстанавливаем регистры

    pop edx

    pop ecx

    pop edi

    pop esi

    pop ebx

    ; Возврат указателя на res

    mov eax, \_res

    ret

sort\_stdcall\_args endp

End DllMain

Файл def для создания библиотеки:

LIBRARY lab2\_c

EXPORTS

\_sort\_stdcall@16 = \_sort\_stdcall@0

\_sort\_cdecl = \_sort\_cdecl@0

@sort\_fastcall@16 = \_sort\_fastcall@0

sort\_cdecl\_args

sort\_stdcall\_args

Функция сортировки на С++, статическое подключение библиотеки и функция тестирования:

#include <iostream>

#include <vector>

#include <assert.h>

#include <chrono>

#pragma comment(lib, "lab2\_c.lib")

extern "C" \_\_declspec(dllimport) long double\* \_stdcall sort\_stdcall(long double\* res, long double\* a, int start, int end);

extern "C" \_\_declspec(dllimport) long double\* \_cdecl sort\_cdecl(long double\* res, long double\* a, int start, int end);

extern "C" \_\_declspec(dllimport) long double\* \_fastcall sort\_fastcall(long double\* res, long double\* a, int start, int end);

extern "C" \_\_declspec(dllimport) long double\* \_stdcall sort\_stdcall\_args(long double\* res, long double\* a, int start, int end);

extern "C" \_\_declspec(dllimport) long double\* \_cdecl sort\_cdecl\_args(long double\* res, long double\* a, int start, int end);

// Функция сортировки пузырьком

long double\* sort(long double\* res, long double\* a, int start, int end) {

// Копируем элементы из исходного массива a в res в диапазоне [start, end]

for (int i = start; i <= end; ++i) {

res[i - start] = a[i];

}

// Сортировка пузырьком

for (int i = start; i <= end; ++i) {

for (int j = start; j < end - (i - start); ++j) {

if (res[j - start] > res[j + 1 - start]) {

// Обмен значениями

double temp = res[j - start];

res[j - start] = res[j + 1 - start];

res[j + 1 - start] = temp;

}

}

}

// Возвращаем указатель на отсортированный массив

return res;

}

template <typename TestedFunction>

void test\_function(TestedFunction func\_to\_test, int amount) {

long double\* a = (long double\*)malloc(sizeof(long double) \* amount);

long double\* res = (long double\*)malloc(sizeof(long double) \* (amount - 2));

for (int i = 0; i < amount; i++) {

a[i] = (long double) (rand() % 1000);

}

std::chrono::steady\_clock::time\_point begin = std::chrono::steady\_clock::now();

func\_to\_test(res, a, 1, amount - 2);

std::chrono::steady\_clock::time\_point end = std::chrono::steady\_clock::now();

auto delta = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin).count();

std::cout << delta / 1000.0 << std::endl;

free(a);

free(res);

}

int main() {

int amount = 1000;

for (size\_t i = 0; i < 10; i++) {

std::cout << "for amount: " << amount << std::endl;

std::cout << " stdcall function: ";

test\_function(sort\_stdcall, amount);

std::cout << " cdecl function: ";

test\_function(sort\_cdecl, amount);

std::cout << " fastcall function: ";

test\_function(sort\_fastcall, amount);

std::cout << " stdcall whit args function: ";

test\_function(sort\_stdcall\_args, amount);

std::cout << " cdecl whit args function: ";

test\_function(sort\_cdecl\_args, amount);

std::cout << " c++ function: ";

test\_function(sort, amount);

std::cout << "\n";

amount += 2500;

}

return 0;

}

Вывод замеров скорости работы функций:

for amount: 1000

stdcall function: 0.001

cdecl function: 0.001

fastcall function: 0.001

stdcall whit args function: 0.001

cdecl whit args function: 0.001

c++ function: 0.001

for amount: 3500

stdcall function: 0.013

cdecl function: 0.012

fastcall function: 0.011

stdcall whit args function: 0.011

cdecl whit args function: 0.011

c++ function: 0.02

for amount: 6000

stdcall function: 0.034

cdecl function: 0.035

fastcall function: 0.037

stdcall whit args function: 0.035

cdecl whit args function: 0.039

c++ function: 0.067

for amount: 8500

stdcall function: 0.086

cdecl function: 0.081

fastcall function: 0.083

stdcall whit args function: 0.078

cdecl whit args function: 0.083

c++ function: 0.127

for amount: 11000

stdcall function: 0.156

cdecl function: 0.154

fastcall function: 0.157

stdcall whit args function: 0.15

cdecl whit args function: 0.158

c++ function: 0.224

for amount: 13500

stdcall function: 0.258

cdecl function: 0.26

fastcall function: 0.259

stdcall whit args function: 0.258

cdecl whit args function: 0.267

c++ function: 0.363

for amount: 16000

stdcall function: 0.402

cdecl function: 0.409

fastcall function: 0.407

stdcall whit args function: 0.407

cdecl whit args function: 0.408

c++ function: 0.537

for amount: 18500

stdcall function: 0.599

cdecl function: 0.609

fastcall function: 0.59

stdcall whit args function: 0.599

cdecl whit args function: 0.608

c++ function: 0.761

for amount: 21000

stdcall function: 0.861

cdecl function: 0.888

fastcall function: 0.87

stdcall whit args function: 0.825

cdecl whit args function: 0.853

c++ function: 1.057

for amount: 23500

stdcall function: 1.108

cdecl function: 1.109

fastcall function: 1.132

stdcall whit args function: 1.108

cdecl whit args function: 1.12

c++ function: 1.356

**Задание на защиту**

Код в стиле fastcall для 64-разрядной системы:

EXTERN printf: PROC

.data

fmt db "%d",10, 0

.code

; double\* sort(double\* res, double\* a, int start, int end).

sort\_fastcall\_args\_64 proc

; Сохранение регистров

push rbx

push rsi

push rdi

push r12

push r13

push r14

push r15

; Извлечение аргументов из регистров

mov rdi, rcx ; Указатель на результирующий массив res (RCX)

mov rsi, rdx ; Указатель на исходный массив a (RDX)

mov rbx, r8 ; Индекс начала start (R8)

mov rcx, r9 ; Индекс конца end (R9)

; Копирование подмассива в res

sub ecx, ebx ; Размер подмассива

inc ecx ; Увеличиваем на 1, чтобы включить end

xor edx, edx ; Индекс в res

copy\_loop:

mov rax, [rsi + rbx \* 8] ; Загрузка элемента из a[start]

mov [rdi + rdx \* 8], rax ; Сохранение в res

inc ebx ; Следующий индекс в a

inc edx ; Следующий индекс в res

loop copy\_loop ; Повторяем для всех элементов

; Инициализация FPU

finit

; Сортировка пузырьком

xor rcx, rcx

mov ecx, r9d

sub ecx, r8d

inc ecx

sort\_outer\_loop:

push rcx ; Сохраняем счетчик внешнего цикла

xor r13, r13 ; Индекс для внутреннего цикла

sort\_inner\_loop:

; Загрузка текущего и следующего элемента

fld qword ptr [rdi + r13 \* 8] ; Загрузка текущего элемента

fld qword ptr [rdi + r13 \* 8 + 8] ; Загрузка следующего элемента

; Сравнение

fcompp ; Сравнение двух чисел в стеке FPU и выталкивание их из стека

fstsw ax ; Сохранение слова состояния FPU в AX

sahf ; Загрузка AH в регистр флагов

; Переход, если текущий элемент <= следующего

jbe no\_swap

; Обмен значениями

fld qword ptr [rdi + r13 \* 8] ; Загрузка текущего элемента

fld qword ptr [rdi + r13 \* 8 + 8] ; Загрузка следующего элемента

fstp qword ptr [rdi + r13 \* 8] ; Сохранение следующего элемента на место текущего

fstp qword ptr [rdi + r13 \* 8 + 8] ; Сохранение текущего элемента на место следующего

no\_swap:

inc r13d ; Следующий индекс

loop sort\_inner\_loop ; Повторяем для всех элементов

pop rcx ; Восстанавливаем счетчик внешнего цикла

loop sort\_outer\_loop ; Повторяем для всех элементов

; Восстановление регистров

pop r15

pop r14

pop r13

pop r12

pop rdi

pop rsi

pop rbx

; Возврат указателя на res

mov rax, rdi

ret

sort\_fastcall\_args\_64 endp

end

Вывод программы для тестов:

for amount: 1000

fastcall\_64 function: 0.002

for amount: 3500

fastcall\_64 function: 0.036

for amount: 6000

fastcall\_64 function: 0.097

for amount: 8500

fastcall\_64 function: 0.189

for amount: 11000

fastcall\_64 function: 0.317

for amount: 13500

fastcall\_64 function: 0.481

for amount: 16000

fastcall\_64 function: 0.676

for amount: 18500

fastcall\_64 function: 0.932

for amount: 21000

fastcall\_64 function: 1.195

for amount: 23500

fastcall\_64 function: 1.499

**Вывод:** Входе лабораторной работы изучены команды поразрядной обработки данных.