**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)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №8**

по дисциплине: Архитектура вычислительных систем

тема: «Способы вызова ассемблерных подпрограмм

в языках высокого уровня»

Выполнил: ст. группы ВТ-221

Беляков Генрих Сергеевич

Проверили:

ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

**Лабораторная работа №8  
Способы вызова ассемблерных подпрограмм**

**в языках высокого уровня  
Вариант 3**

**Цель работы:** изучение команд поразрядной обработки данных.

**Задания для выполнения к работе:**

1. Написать и отладить подпрограммы на masm32 в разных стилях вызова для решения задачи соответствующего варианта. Глобальные переменные в подпрограммах использовать не разрешается. Если нужна дополнительная память, выделять её в стеке.
2. Подпрограммы собрать и скомпилировать в виде dll-библиотеки. Библиотека должна содержать:
   1. подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере без явного перечисления аргументов в заголовке;
   2. Подпрограммы в стилях stdcall, cdecl, написанные, наоборот, с перечислением аргументов в заголовке подпрограммы.
3. Подключить все подпрограммы из dll-библиотеки к проектам на C# и С++ статическим и динамическим способом. Убедиться в правильности вызова всех подпрограмм.
4. Написать подпрограмму для решения задачи варианта с использованием ассемблерной вставки на языке C++.
5. Написать подпрограммы для решения задачи варианта с использованием обычного высокоуровнего языка C# и C++ (или любого другого).
6. Сравнить скорость выполнения полученных подпрограмм на одних и тех же тестовых данных. Для сравнения выбрать: подпрограмму на ассемблере в masm32 (какую-нибудь одну из пяти), вызываемую из программы на языке C++ или C#; подпрограмму на C#; подпрограмму на C++; подпрограмму на С++ с использованием ассемблерной вставки. Построить на одной плоскости графики зависимости времени выполнения подпрограмм от длины массивов (не менее 10 точек для каждой подпрограммы). Для замера лучше передавать в подпрограммы массивы большой длины. Время замерять в миллисекундах с помощью API-функции GetTickCount(). Проверить, что подпрограммы при одинаковых тестовых данных выдают одинаковый результат. Для заполнения массивов использовать генератор случайных чисел.
7. В отчёт включить весь исходный код и графики.
8. Сделать выводы по работе.

**Задание:**

|  |  |
| --- | --- |
| 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. | |
| 3 | Сортировка выбором по не убыванию.  int\* sort(int\* a, int start, int end, int\* res). |

Исходный код (asm):

.686

.model flat, stdcall

option casemap: none

include windows.inc

include kernel32.inc

include msvcrt.inc

includelib  kernel32.lib

includelib  msvcrt.lib

.code

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

mov eax, 1

ret

DllMain endp

; select\_sort(int\* a, int length)

select\_sort proc

    pushad

    mov    ebp, esp

    mov    ecx, [ebp + 32 + 4 + 4] ; Количество чисел

    mov    ebx, [ebp + 32 + 4] ; Адрес числа

    sub    ecx, 1       ; Готово если 0 или 1 элемент

    jbe    select\_sort\_end

select\_sort\_outer\_loop:

        mov    edx, ecx      ; Количество сравнений

        mov    esi, ebx      ; Начало неотсортированного массива

        mov    eax, [esi]    ; Минимум (число)

        mov    edi, esi      ; Минимум (адрес)

select\_sort\_inner\_loop:

            add    esi, 4

            cmp [esi], eax

            jg     select\_sort\_not\_smaller

            mov    eax, [esi]    ; Значение минимума

            mov    edi, esi      ; Адрес минимума

select\_sort\_not\_smaller:

            dec    edx

            jnz    select\_sort\_inner\_loop

        mov    edx, [ebx]    ; Обмен элементов

        mov    [ebx], eax

        mov    [edi], edx

        add    ebx, 4       ; Передвинуть границу отсортированного массива

        dec    ecx

        jnz    select\_sort\_outer\_loop

select\_sort\_end:

    popad

    ret    8

select\_sort endp

; int\* sort\_stdcall\_noarg(int\* a, int start, int end, int\* res)

sort\_stdcall\_noarg proc

    push ebp

    push edi

    push esi

    push ebx

    mov ecx, [esp + 20]       ; ecx = a

    mov edx, [esp + 20 + 4]   ; edx = start

    mov edi, [esp + 20 + 8]   ; edi = end

    mov ebx, [esp + 20 + 12]  ; ebx = res

    mov eax, 0

    ; Копируем данные в res

sort\_stdcall\_noarg\_copyloop:

    ; start > end?

    cmp edx, edi

    jg sort\_stdcall\_noarg\_copyloop\_end

    ; ebp = res + eax \* 4

    mov ebp, ebx

    add ebp, eax

    add ebp, eax

    add ebp, eax

    add ebp, eax

    push edi

    push eax

    ; edi = a + start \* 4

    mov edi, ecx

    add edi, edx

    add edi, edx

    add edi, edx

    add edi, edx

    ; res[eax] = a[start]

    mov eax, dword ptr [edi]

    mov dword ptr [ebp], eax

    pop eax

    pop edi

    ; start++

    ; eax++

    inc eax

    inc edx

    jmp sort\_stdcall\_noarg\_copyloop

sort\_stdcall\_noarg\_copyloop\_end:

    push eax

    push ebx

    call select\_sort

    mov eax, ebx

    pop ebx

    pop esi

    pop edi

    pop ebp

    ret 4 \* 4

sort\_stdcall\_noarg endp

; int sort\_cdecl\_noarg (int\* a, int start, int end, int\* res)

sort\_cdecl\_noarg proc

    push ebp

    push edi

    push esi

    push ebx

    mov ecx, [esp + 20]       ; ecx = a

    mov edx, [esp + 20 + 4]   ; edx = start

    mov edi, [esp + 20 + 8]   ; edi = end

    mov ebx, [esp + 20 + 12]  ; ebx = res

    mov eax, 0

    ; Копируем данные в res

sort\_cdecl\_noarg\_copyloop:

    ; start > end?

    cmp edx, edi

    jg sort\_cdecl\_noarg\_copyloop\_end

    ; ebp = res + eax \* 4

    mov ebp, ebx

    add ebp, eax

    add ebp, eax

    add ebp, eax

    add ebp, eax

    push edi

    push eax

    ; edi = a + start \* 4

    mov edi, ecx

    add edi, edx

    add edi, edx

    add edi, edx

    add edi, edx

    ; res[eax] = a[start]

    mov eax, dword ptr [edi]

    mov dword ptr [ebp], eax

    pop eax

    pop edi

    ; start++

    ; eax++

    inc eax

    inc edx

    jmp sort\_cdecl\_noarg\_copyloop

sort\_cdecl\_noarg\_copyloop\_end:

    push eax

    push ebx

    call select\_sort

    mov eax, ebx

    pop ebx

    pop esi

    pop edi

    pop ebp

    ret

sort\_cdecl\_noarg endp

; int sort\_fastcall\_noarg(int\* a, int start, int end, int\* res)

sort\_fastcall\_noarg proc

    push ebp

    push edi

    push esi

    push ebx

    mov edi, [esp + 20]   ; edi = end

    mov ebx, [esp + 20 + 4]  ; ebx = res

    mov eax, 0

    ; Копируем данные в res

sort\_fastcall\_noarg\_copyloop:

    ; start > end?

    cmp edx, edi

    jg sort\_fastcall\_noarg\_copyloop\_end

    ; ebp = res + eax \* 4

    mov ebp, ebx

    add ebp, eax

    add ebp, eax

    add ebp, eax

    add ebp, eax

    push edi

    push eax

    ; edi = a + start \* 4

    mov edi, ecx

    add edi, edx

    add edi, edx

    add edi, edx

    add edi, edx

    ; res[eax] = a[start]

    mov eax, dword ptr [edi]

    mov dword ptr [ebp], eax

    pop eax

    pop edi

    ; start++

    ; eax++

    inc eax

    inc edx

    jmp sort\_fastcall\_noarg\_copyloop

sort\_fastcall\_noarg\_copyloop\_end:

    push eax

    push ebx

    call select\_sort

    mov eax, ebx

    pop ebx

    pop esi

    pop edi

    pop ebp

    ret 2 \* 4

sort\_fastcall\_noarg endp

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

sort\_stdcall proc stdcall a: DWORD, index\_start: DWORD, index\_end: DWORD, res: DWORD

    push edi

    push esi

    push ebx

    mov eax, 0

    ; Копируем данные в res

sort\_stdcall\_copyloop:

    ; start > end?

    mov esi, index\_start

    cmp esi, index\_end

    jg sort\_stdcall\_copyloop\_end

    ; ebx = res + eax \* 4

    mov ebx, res

    add ebx, eax

    add ebx, eax

    add ebx, eax

    add ebx, eax

    push edi

    push eax

    ; edi = a + start \* 4

    mov edi, a

    add edi, index\_start

    add edi, index\_start

    add edi, index\_start

    add edi, index\_start

    ; res[eax] = a[start]

    mov eax, dword ptr [edi]

    mov dword ptr [ebx], eax

    pop eax

    pop edi

    ; start++

    ; eax++

    inc eax

    inc index\_start

    jmp sort\_stdcall\_copyloop

sort\_stdcall\_copyloop\_end:

    push eax

    push res

    call select\_sort

    mov eax, res

    pop ebx

    pop esi

    pop edi

    ret

sort\_stdcall endp

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

sort\_cdecl proc c a: DWORD, index\_start: DWORD, index\_end: DWORD, res: DWORD

    push edi

    push esi

    push ebx

    mov eax, 0

    ; Копируем данные в res

sort\_cdecl\_copyloop:

    ; start > end?

    mov esi, index\_start

    cmp esi, index\_end

    jg sort\_cdecl\_copyloop\_end

    ; ebx = res + eax \* 4

    mov ebx, res

    add ebx, eax

    add ebx, eax

    add ebx, eax

    add ebx, eax

    push edi

    push eax

    ; edi = a + start \* 4

    mov edi, a

    add edi, index\_start

    add edi, index\_start

    add edi, index\_start

    add edi, index\_start

    ; res[eax] = a[start]

    mov eax, dword ptr [edi]

    mov dword ptr [ebx], eax

    pop eax

    pop edi

    ; start++

    ; eax++

    inc eax

    inc index\_start

    jmp sort\_cdecl\_copyloop

sort\_cdecl\_copyloop\_end:

    push eax

    push res

    call select\_sort

    mov eax, res

    pop ebx

    pop esi

    pop edi

    ret

sort\_cdecl endp

end DllMain

libs.def:

LIBRARY libs

EXPORTS

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

\_sort\_cdecl\_noarg = \_sort\_cdecl\_noarg@0

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

sort\_cdecl

sort\_stdcall

Исходный тестирующий код (C++):

#include <iostream>

#include <vector>

#include <assert.h>

#include <chrono>

#pragma comment(lib, "libs.lib")

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

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

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

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

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

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

    int i = 0;

    int length = 0;

    while (start <= end) {

        res[i++] = a[start++];

        length++;

    }

    if (length <= 1) return res;

    for (i = 0; i < length - 1; i++) {

        int min\_element\_index = i;

        for (int j = i + 1; j < length; j++)

            if (res[j] < res[min\_element\_index])

                min\_element\_index = j;

        int tmp = res[min\_element\_index];

        res[min\_element\_index] = res[i];

        res[i] = tmp;

    }

    return res;

}

template <typename TestedFunction>

void test\_function1(TestedFunction func\_to\_test) {

    int a[] = { -1, -2, -3, -4, -5, -6 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == -6 &&

        sort\_res[1] == -5 &&

        sort\_res[2] == -4 &&

        sort\_res[3] == -3 &&

        sort\_res[4] == -2 &&

        sort\_res[5] == -1);

}

template <typename TestedFunction>

void test\_function2(TestedFunction func\_to\_test) {

    int a[] = { 6, 5, 4, 3, 2, 1 };

    int sort\_res[4] = {};

    auto res = func\_to\_test(a, 1, 4, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == 2 &&

        sort\_res[1] == 3 &&

        sort\_res[2] == 4 &&

        sort\_res[3] == 5);

}

template <typename TestedFunction>

void test\_function3(TestedFunction func\_to\_test) {

    int a[] = { -1, 2, -3, 3, 45, -6 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == -6 &&

        sort\_res[1] == -3 &&

        sort\_res[2] == -1 &&

        sort\_res[3] == 2 &&

        sort\_res[4] == 3 &&

        sort\_res[5] == 45);

}

template <typename TestedFunction>

void test\_function4(TestedFunction func\_to\_test) {

    int a[] = { -6, -5, -4, -3, -2, -1 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == -6 &&

        sort\_res[1] == -5 &&

        sort\_res[2] == -4 &&

        sort\_res[3] == -3 &&

        sort\_res[4] == -2 &&

        sort\_res[5] == -1);

}

template <typename TestedFunction>

void test\_function5(TestedFunction func\_to\_test) {

    int a[] = { 1, 2, 3, 4, 5, 6 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == 1 &&

        sort\_res[1] == 2 &&

        sort\_res[2] == 3 &&

        sort\_res[3] == 4 &&

        sort\_res[4] == 5 &&

        sort\_res[5] == 6);

}

template <typename TestedFunction>

void test\_function6(TestedFunction func\_to\_test) {

    int a[] = { 6, 3, 4, 2, 1, 5 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == 1 &&

        sort\_res[1] == 2 &&

        sort\_res[2] == 3 &&

        sort\_res[3] == 4 &&

        sort\_res[4] == 5 &&

        sort\_res[5] == 6);

}

template <typename TestedFunction>

void test\_function7(TestedFunction func\_to\_test) {

    int a[] = { -4, -2, -6, -1, -5, -3 };

    int sort\_res[6] = {};

    auto res = func\_to\_test(a, 0, 5, sort\_res);

    assert(res == sort\_res);

    assert(

        sort\_res[0] == -6 &&

        sort\_res[1] == -5 &&

        sort\_res[2] == -4 &&

        sort\_res[3] == -3 &&

        sort\_res[4] == -2 &&

        sort\_res[5] == -1);

}

template <typename TestedFunction>

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

    srand(0);

    int \*a = (int\*)malloc(sizeof(int) \* amount);

    int \*sort\_res = (int\*)malloc(sizeof(int) \* amount);

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

        a[i] = rand() % 1000;

    }

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

    auto res = func\_to\_test(a, 0, amount - 1, sort\_res);

    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 << "Working time: " << delta / 1000.0 << std::endl;

    free(a);

    free(sort\_res);

}

template <typename TestedFunction>

void test\_function(TestedFunction func\_to\_test) {

    test\_function1(func\_to\_test);

    test\_function2(func\_to\_test);

    test\_function3(func\_to\_test);

    test\_function4(func\_to\_test);

    test\_function5(func\_to\_test);

    test\_function6(func\_to\_test);

    test\_function7(func\_to\_test);

    for (int i = 10000; i <= 25000; i += 1000) {

        stress\_test(func\_to\_test, i);

    }

}

int main() {

    std::cout << "Native function:" << std::endl;

    test\_function(sort\_native);

    std::cout << "\_\_cdecl auto parameters:" << std::endl;

    test\_function(sort\_cdecl);

    std::cout << "\_\_stdcall auto parameters:" << std::endl;

    test\_function(sort\_stdcall);

    std::cout << "\_\_stdcall manual parameters" << std::endl;

    test\_function(sort\_stdcall\_noarg);

    std::cout << "\_\_cdecl manual parameters" << std::endl;

    test\_function(sort\_cdecl\_noarg);

    std::cout << "\_\_fastcall manual parameters:" << std::endl;

    test\_function(sort\_fastcall\_noarg);

    return 0;

}

Графики времени выполнения:

Оранжевый – время выполнения для вручную написанного кода, синий – релизная версия функции сортировки, скомпилированная средствами Visual Studio 2022.

**Вывод:** в ходе лабораторной изучили способы вызова подпрограмм, написанных на разных языках программирования посредством dll-библиотек. В большинстве случаев скомпилированный код будет быстрее и надёжнее кода, написанного вручную. Время создания ассемблерного кода неоправданно гораздо больше времени создания кода, разработанного при помощи инструментов языков программирования высокого уровня.