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:
    a. подпрограммы в стилях stdcall, cdecl, fastcall, написанные на ассемблере без явного перечисления аргументов в заголовке;
    b. 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 C++ 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 C++ 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:

<table>
<tr><td colspan="2">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.</td></tr>
<tr><td>3</td><td>Сортировка выбором по не убыванию.
<code>int* sort(int* a, int start, int end, int* res).</code></td></tr>
</table>

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++):

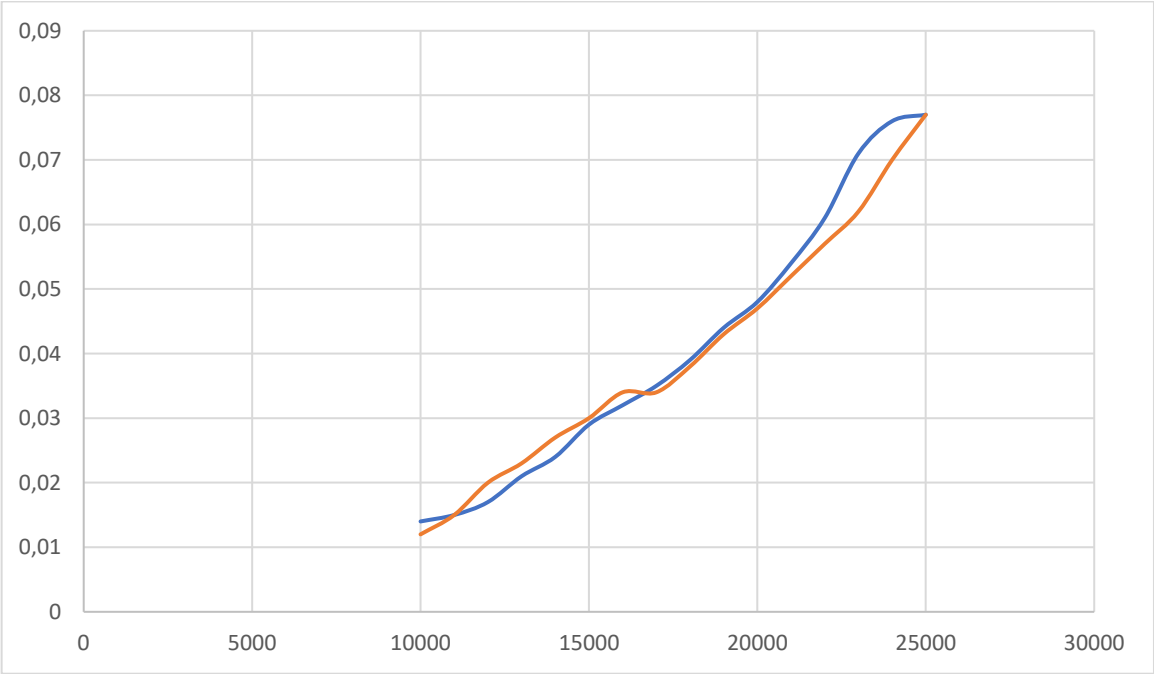
```
#include <iostream>
#include <vector>
#include <assert.h>
#include <chrono>

#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.