Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

# Системне програмування

# Лабораторна робота №9

«Використання у проекті C++ модулів на Асемблері»

Виконав:
студент групи ІО-24
Довгань М. С.

Перевірив:
Порєв В. М.

Київ - 2024

**Тема:** Використання у проекті C++ модулів на Асемблері.

**Мета:** Навчитися програмувати модулі на асемблері, у яких містяться команди SSE, x87 FPU а також використовувати такі модулі у проектах C++.

**Завдання:**

1. Створити проект Windows Desktop Application з ім'ям Lab9.
2. Написати на асемблері процедуру обчислення скалярного добутку двох векторів із використанням команд SSE. Ім'я процедури: **MyDotProduct_SSE**. Процедуру оформити у окремому модулі і записати файли vectsse.asm, vectsse.h. Додати файл vectsse.asm у проект.
3. Запрограмувати на асемблері процедуру обчислення скалярного добутку двох векторів на основі команд x87 FPU без використання команд SSE. Ім'я процедури: **MyDotProduct_FPU**. Процедуру оформити у окремому модулі і записати файли vectfpu.asm, vectfpu.h. Додати файл vectfpu.asm у проект.
4. Запрограмувати на C++ обчислення скалярного добутку тих самих векторів як звичайну функцію C++ з ім'ям **MyDotProduct**, яка приймає значення двох масивів і записує результат у числову перемінну (будь-яка оптимізація при компіляції повинна бути відсутня).
5. Зробити меню для вікна програми так, щоб користувач програми мав можливість викликати процедури на асемблері MyDotProduct_SSE, MyDotProduct_FPU з модулів vectsse, vectfpu, а також функцію MyDotProduct.
6. Запрограмувати вивід результатів обчислень та виміри часу виконання скалярного добутку для трьох варіантів реалізації.
7. Отримати дизасемблерний текст функції C++ MyDotProduct. Проаналізувати код дизасемблеру, порівняти з кодом на асемблері процедури MyDotProduct_FPU.
8. Зробити висновки щодо використання модулів на асемблері у проектах програм, створених на основі мови C++ .

**Індивідуальний варіант завдання:**

1. У середовищі MS Visual Studio створити новий проект C++ з ім’ям **Lab9**.

2. Запрограмувати у модулях на асемблері процедури обчислення скалярного добутку. Процедури двох видів: **MyDotProduct_FPU(&res, A, B, N)** на основі команд x87 FPU **MyDotProduct_SSE(&res, A, B, N)** на основі команд SSE Також треба запрограмувати вже мовою C/C++ третю процедуру (функцію) для обчислення скалярного добутку **MyDotProduct(&res, A, B, N)**; У головному файлі Lab9 мають бути виклики трьох функцій. Вибір – через меню.

3. Кількість елементів векторів A та B має бути N = 480 × (номер студента у журналі), підставимо та отримаємо кількість елементів векторіа A та B: N = 480 × 9 = 4320.

4. Запрограмувати знаходження часу виконання для кожного з варіантів обчислення скалярного добутку.

5. Програма повинна виводити числове значення скалярного добутку та час виконання у вікні MessageBox – окремо для кожного з варіантів реалізації.

**Виконання завдання:**

**Роздруківка коду програми:**

vectsse.h:

```
#pragma once
extern "C"
{
	void MyDotProduct_SSE(float* dest, float* pB, float* pA, long bits);
}
```

vectfpu.h:

```
#pragma once
extern "C"
{
	void MyDotProduct_FPU(float* dest, float* pB, float* pA, long bits);
}
```

vectsse.asm:

```
.686
.xmm
.model flat, C
.data

temp dd 4 dup(0)

.code
MyDotProduct_SSE proc dest:DWORD, pB:DWORD, pA:DWORD, bits:DWORD ; Початок процедури MyDotProduct_SSE з чотирма параметрами

	mov edx, bits
	mov esi, pA
	mov ebx, pB
	mov edi, dest

	cycle:
		sub edx, 4
		movups xmm0, [esi+edx*4]
		movups xmm1, [ebx+edx*4]
		mulps xmm0, xmm1
		addps xmm2, xmm0
		cmp edx, 0
	jne cycle

	haddps xmm2, xmm2
	haddps xmm2, xmm2
	movups temp, xmm2
	mov eax, dword ptr[temp]
	mov dword ptr[edi], eax
```

```
	ret

MyDotProduct_SSE endp

end
```

## vectfpu.asm:

```
.686
.xmm
.model flat, c

.data

temp dd 4 dup(0)

.code

MyDotProduct_FPU proc dest:DWORD, pB:DWORD, pA:DWORD, bits:DWORD ; Початок процедури MyDotProduct_FPU з чотирма параметрами

	mov edx, bits
	mov esi, pA
	mov ebx, pB
	mov edi, dest
	fld dword ptr[temp]

	cycle:
		dec edx
		fld dword ptr[esi+edx*4]
		fmul dword ptr[ebx+edx*4]
		faddp st(1), st(0)
		cmp edx, 0
	jne cycle
	fstp dword ptr[edi]
	ret

MyDotProduct_FPU endp
```

```
end
```

## Lab9.cpp:

```
// Lab9.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include <cstdlib>
#include <string>
#include "Lab9.h"

#include "vectsse.h"
#include "vectfpu.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class
name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

void myWorkSSE(HWND hWnd);
void myWorkFPU(HWND hWnd);
void myWorkCPP(HWND hWnd);


int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
```

```
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB9, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB9));

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
```

```
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB9));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_LAB9);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global
variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable
```

```
   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance,
nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
            case ID_TASK_SSE:
                myWorkSSE(hWnd);
                break;
```

```
            case ID_TASK_FPU:
                myWorkFPU(hWnd);
                break;
            case ID_TASK_C:
                myWorkCPP(hWnd);
                break;

            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd,
About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code that uses hdc here...
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
```

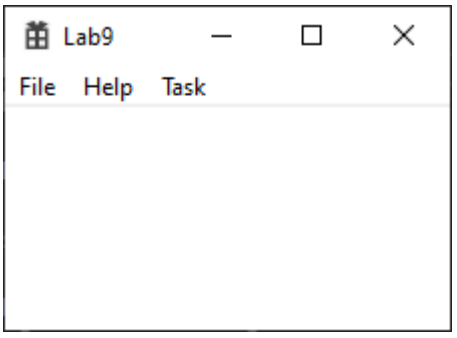
```
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
void myWorkSSE(HWND hWnd)
{
    float* arrayA = new float[4320];
    float* arrayB = new float[4320];
    float result0 = 0;

    for (int i = 0; i < 4320; i++) arrayA[i] = 0.001f * (float)(i + 1);
    for (int i = 0; i < 4320; i++) arrayB[i] = 0.001f * (float)(i + 1);

    MyDotProduct_SSE(&result0, arrayB, arrayA, 4320);


    std::string text = std::to_string(result0);
    MessageBoxA(hWnd, text.c_str(), "Скалярний добуток SSE", MB_OK);

    SYSTEMTIME st;
    long tst, ten;
    GetLocalTime(&st);
    tst = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds;

    for (long i = 0; i < 1000000; i++)
```

```
    {
        MyDotProduct_SSE(&result0, arrayB, arrayA, 4320);
    }

    GetLocalTime(&st);
    ten = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds - tst;

    delete[]arrayA;
    delete[]arrayB;
    std::string text2 = std::to_string(ten);
    MessageBoxA(hWnd, text2.c_str(), "Час виконання SSE", MB_OK);

}

void myWorkFPU(HWND hWnd)
{
    float* arrayA = new float[4320];
    float* arrayB = new float[4320];
    float result0 = 0;

    for (int i = 0; i < 4320; i++) arrayA[i] = 0.001f * (float)(i + 1);
    for (int i = 0; i < 4320; i++) arrayB[i] = 0.001f * (float)(i + 1);

    MyDotProduct_FPU(&result0, arrayB, arrayA, 4320);



    std::string text = std::to_string(result0);
    MessageBoxA(hWnd, text.c_str(), "Скалярний добуток FPU", MB_OK);

    SYSTEMTIME st;
    long tst, ten;
    GetLocalTime(&st);
    tst = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds;

    for (long i = 0; i < 1000000; i++)
    {
```

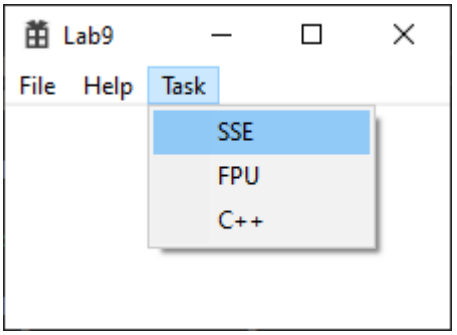
```
        MyDotProduct_FPU(&result0, arrayB, arrayA, 4320);
    }
    GetLocalTime(&st);
    ten = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds - tst;

    delete[]arrayA;
    delete[]arrayB;
    std::string text2 = std::to_string(ten);
    MessageBoxA(hWnd, text2.c_str(), "Час виконання FPU", MB_OK);
}

void myWorkCPP(HWND hWnd)
{
    float* arrayA = new float[4320];
    float* arrayB = new float[4320];
    float result = 0;

    for (int i = 0; i < 4320; i++) arrayA[i] = 0.001f * (float)(i + 1);
    for (int i = 0; i < 4320; i++) arrayB[i] = 0.001f * (float)(i + 1);

    SYSTEMTIME st;
    long tst, ten;
    GetLocalTime(&st);
    tst = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds;

    for (long i = 0; i < 1000000; i++)
    {
        result = 0;
        for (long i = 0; i < 4320; i++)
        {
            result = result + arrayA[i] * arrayB[i];
        }

    }

    GetLocalTime(&st);
```

```
    ten = 60000 * (long)st.wMinute + 1000 * (long)st.wSecond +
(long)st.wMilliseconds - tst;
    delete[]arrayA;
    delete[]arrayB;

    std::string text = std::to_string(result);
    MessageBoxA(hWnd, text.c_str(), "Скалярний добуток C++", MB_OK);

    std::string text3 = std::to_string(ten);
    MessageBoxA(hWnd, text3.c_str(), "Час виконання C++", MB_OK);
}
```

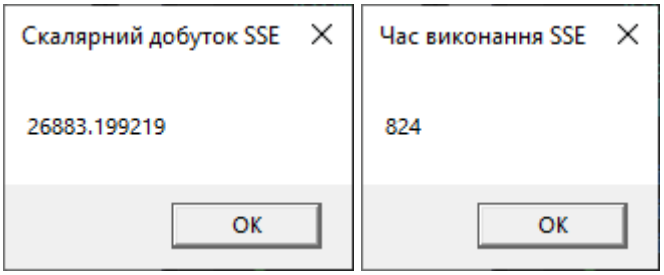
**Результати виконання програми:**



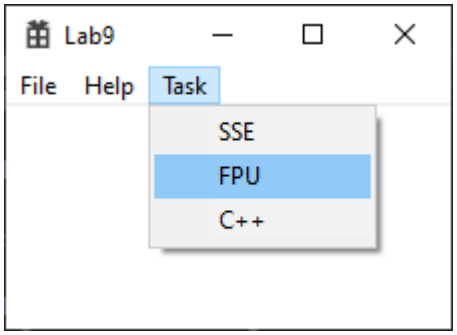

Натискаємо на випадне меню Task:



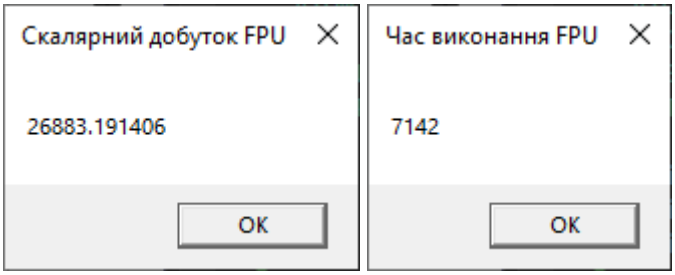

Обираємо SSE:



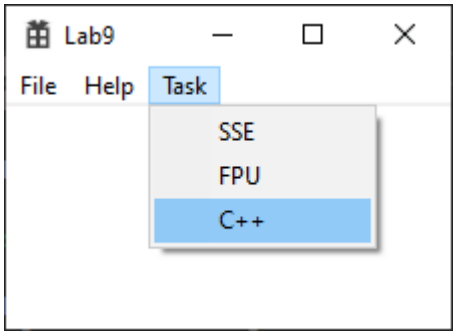
Натискаємо на випадне меню Task:



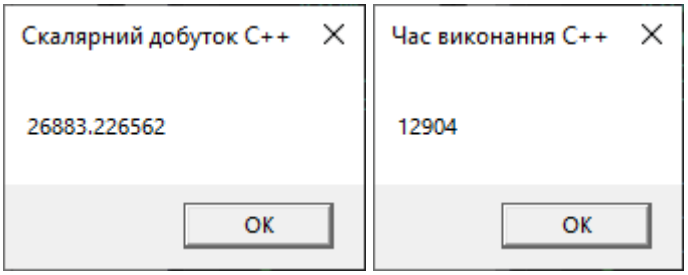

Обираємо FPU:




Натискаємо на випадне меню Task:




Обираємо C++:



**Аналіз виконання програми:**

Створена мною програма виконує завдання лабораторної роботи, відповідно до мого індивідуального варіанту завдання. При запуску програми ми можемо побачити перед собою вікно нашого проекту, в якому містяться випадні вікна - File, Help та Task. Нам необхідне лише вікно Task, ми натискаємо на нього і обираємо SSE - після чого в окремому вікні нам виводиться скалярний добуток SSE, у моєму випадку, це значення 26883.199219. Після закриття цього вікна нам видає нове, в якому вказаний час виконання SSE, який становить 841. Ту ж саму процедуру ми пророблюємо для FPU та C++. Скалярний добуток FPU становить 26883.19140, а час його виконання - 7156, що, небагато, немало - але майже в 10 разів перевищує значення за SSE. Обираємо останнє завдання - C++, його скалярний добуток обчислений в 26883.226562, а час виконання - 12968, що майже у два рази більше за FPU та в 15 разів! більше за час у SSE. З цих даних можна зрозуміти, що час виконання SSE є найшвидшим, після нього йде FPU, а останнє місце в цій трійці займає C++.

**Висновок:** під час виконання даної лабораторної роботи я навчився програмувати модулі на Асемблері, в яких містяться команди SSE, x87 FPU, а також використовував такі модулі у проектах C++.