Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №11**

з дисципліни «Системне програмування»

Виконав:

студент групи ІО-32

Попенко Р.Л.

Перевірив:

Порєв М.В.

Київ, 2015 р.

**Завдання:**

1. Створити проект Visual C++ Win32 з ім'ям Lab11.

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 програмах на

мові C++ .

**N=40\*24=960**

**Код програми**

***Lab11.cpp (частина)***

const int small = 8;

const int huge = 960;

int n = 120;

float a0[small] = { 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 18.0 };

float b0[small] = { 18.0, 14.0, 12.0, 10.0, 8.0, 5.0, 3.0, 1.0 };

float a[huge];

float b[huge];

float dest[1]= { 0 };

char texbuf[128];

char decCode[32];

void firstVecSSE(HWND hWnd) {

forming();

SYSTEMTIME st;

long tst;

long ten;

GetLocalTime(&st);

tst = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds;

for (long i = 0; i<10000000; i++)

{

vectorSSE(dest, a, b, huge\*4);

}

GetLocalTime(&st);

ten = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds - tst;

BinToDec\_Module(dest, texbuf);

MessageBox(hWnd, texbuf, "Результат SSE", MB\_OK);

StrToDec\_LONGOP(5, 1, decCode, ten);

MessageBox(hWnd, decCode, "час виконання SSE", MB\_OK);

}

void firstVecFPU(HWND hWnd) {

forming();

SYSTEMTIME st;

long tst;

long ten;

GetLocalTime(&st);

tst = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds;

for (long i = 0; i<10000000; i++)

{

vectorFPU(dest, a, b, huge\*4); }

GetLocalTime(&st);

ten = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds - tst;

BinToDec\_Module(dest, texbuf);

MessageBox(hWnd, texbuf, "Результат FPU", MB\_OK);

StrToDec\_LONGOP(5, 1, decCode, ten);

MessageBox(hWnd, decCode, "час виконання FPU", MB\_OK);

}

void cppVec(HWND hWnd) {

forming();

SYSTEMTIME st;

long tst;

long ten;

GetLocalTime(&st);

tst = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds;

for (long i = 0; i<10000000; i++)

{

skal(a, b, dest); }

GetLocalTime(&st);

ten = 60000 \* (long)st.wMinute

+ 1000 \* (long)st.wSecond

+ (long)st.wMilliseconds - tst;

BinToDec\_Module(dest, texbuf);

MessageBox(hWnd, texbuf, "Результат C++", MB\_OK);

StrToDec\_LONGOP(5, 1, decCode, ten);

MessageBox(hWnd, decCode, "час виконання C++", MB\_OK);

}

void skal(float \*a, float \*b, float \*dest){

dest[0] = 0;

for (int j = 0; j < 760; j++)

dest[0] = dest[0] + a[j] \* b[j];

}

void forming(){

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

{

for (int j = 0; j < small; j++)

{

a[small\*i + j] = a0[j];

}

}

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

{

for (int j = 0; j < small; j++)

{

b[small\*i + j] = b0[j];

}

}

}

***Module.ASM***

.586

.model flat, c

.data

base dd 0

tmpEBX dd 0

resAddr dd 0

tmp dd 10 dup(0)

expon dt 0

mant dt 0

x dd 0h

fractPart db ?

x1 dd 0h

x2 dd 0h

b dd 0

two dd 2

decCode db ?

buf dd 80 dup(0)

.code

BinToDec\_Module proc src: dword, result: dword

mov eax, src;[ebp+16] ; число

mov ebx, result;[ebp+12] ; результат

xor ecx, ecx

xor edx, edx

mov cx, word ptr [eax+2]

shr cx, 7

sub cl, 127

mov edx, dword ptr [eax]

mov dword ptr [expon], edx

and word ptr [expon+2], 0000000001111111b

or word ptr [expon+2], 0000000010000000b

mov edx, dword ptr [expon]

mov dword ptr [mant], edx

mov ch, cl

@cycle1:

shl dword ptr [mant], 1

and dword ptr [mant+2], 0000000001111111b

dec ch

cmp ch, 0

jg @cycle1

mov ch, 23

sub ch, cl

mov cl, ch ; експонента

shr dword ptr [expon], cl

mov byte ptr [ebx], 02Eh

mov esi, 0

@decWrite:

mov edx, dword ptr [mant]

shl dword ptr [mant], 1

shl edx, 3

add dword ptr [mant], edx

mov dx, 0

or dx, 0000011110000000b

and dx, word ptr [mant+2]

and word ptr [mant+2], 0000000001111111b

shr dx, 7

add dl, 48

mov byte ptr [ebx+esi+1], dl

inc esi

cmp esi, 3

jl @decWrite

push offset expon

push 8

push ebx

call ToDecStr\_Module

ret ;12

BinToDec\_Module endp

ToDecStr\_Module proc

push ebp

mov ebp,esp

mov esi, [ebp+16]; число

mov ebx, [ebp+12]; розмір числа в байтах

mov ecx, [ebp+8] ; результат

sub ebx, 4

mov base, 10

mov tmpEBX, ebx

mov edi, 10

@moreCycles:

mov ebx, tmpEBX

mov edx, 0

@cycle1:

mov eax, dword ptr [esi+ebx]

div edi

mov dword ptr [esi+ebx], 0

mov dword ptr [esi+ebx], eax

sub ebx, 4

cmp ebx, 0

jge @cycle1

add dl, 48 ; magic number

mov al, byte ptr [ecx+7]

mov byte ptr [ecx+8], al

shl dword ptr [ecx+4], 8

mov al, byte ptr [ecx+3]

mov byte ptr [ecx+4], al

shl dword ptr [ecx], 8

mov byte ptr [ecx], dl

mov edx, tmpEBX

@checkCycle:

sub edx, 4

cmp dword ptr [esi+edx], 0

jne @moreCycles

cmp edx, 0

jg @checkCycle

pop ebp

ret 12

ToDecStr\_Module endp

DIV2\_LONGOP proc

push ebp

mov ebp, esp

mov esi, [ebp + 20] ; number

mov edi, [ebp + 16] ;integer

mov ebx, [ebp + 12] ;fractional

mov eax, [ebp + 8] ; bytes

mov x, eax

push ebx

xor edx, edx

mov ecx, x

dec x

mov ebx,x

@cycle :

push ecx

mov ecx, 10

mov eax, esi

div ecx

mov fractPart, dl

mov dword ptr[edi + 4 \* ebx], eax

dec ebx

pop ecx

dec ecx

jnz @cycle

pop ebx

mov al, fractPart

mov byte ptr[ebx], al

pop ebp

ret 16

DIV2\_LONGOP endp

StrToDec\_LONGOP proc bytesOnScreen: dword, numberOfDd: dword, decCodeLocal: dword, strCodeLocal: dword

mov esi, strCodeLocal

mov edi, decCodeLocal

mov eax, numberOfDd

mov x1, eax

mov eax, bytesOnScreen

mov x2, eax

mov b, 0

xor ecx, ecx

xor ebx, ebx

@cycle:

push ecx

push edi

push esi

push offset buf

push offset decCode

push x1

call DIV2\_LONGOP

pop edi

mov ebx, b

mov al, byte ptr[decCode]

add al, 48

mov byte ptr[edi + ebx], al

xor ecx, ecx

@cycleInner:

mov eax, dword ptr[buf + 4 \* ecx]

mov esi, eax

mov dword ptr[buf + 4 \* ecx], 0

inc ecx

cmp ecx, x1

jl @cycleInner

pop ecx

inc ecx

inc b

cmp ecx, x2

jl @cycle

mov ebx, x2

mov eax, x2

xor edx, edx

div two

mov x2, eax

dec ebx

xor ecx, ecx

@cycle1:

mov al, byte ptr[edi + ecx]

mov ah, byte ptr[edi + ebx]

mov byte ptr[edi + ecx], ah

mov byte ptr[edi + ebx], al

dec ebx

inc ecx

cmp ecx, x2

jl @cycle1

;add esp, 16

ret

StrToDec\_LONGOP endp

end

***Vector.ASM***

.686

.xmm

.model flat, c

.data

null dd 4 dup(0)

res dd 0

.code

vectorSSE proc dest: dword, K: dword, L: dword, num: dword

mov ecx, num

mov edx, L

mov eax, K

mov edi, dest

movups xmm2, [null]

@cycle:

sub ecx, 16

movups xmm0, [eax + ecx]

movups xmm1, [edx + ecx]

mulps xmm0, xmm1

addps xmm2, xmm0

cmp ecx, 0

jne @cycle

haddps xmm2,xmm2

haddps xmm2,xmm2

movss dword ptr[edi], xmm2

;add esp, 12

ret

vectorSSE endp

vectorFPU proc dest: dword, K: dword, L: dword, num: dword

mov ecx, num

mov edi, dest

mov eax, K

mov edx, L

fld res

@cycle:

sub ecx, 4

fld dword ptr[eax + ecx]

fmul dword ptr[edx + ecx]

faddp st(1), st(0)

cmp ecx, 0

jne @cycle

fstp dword ptr[edi]

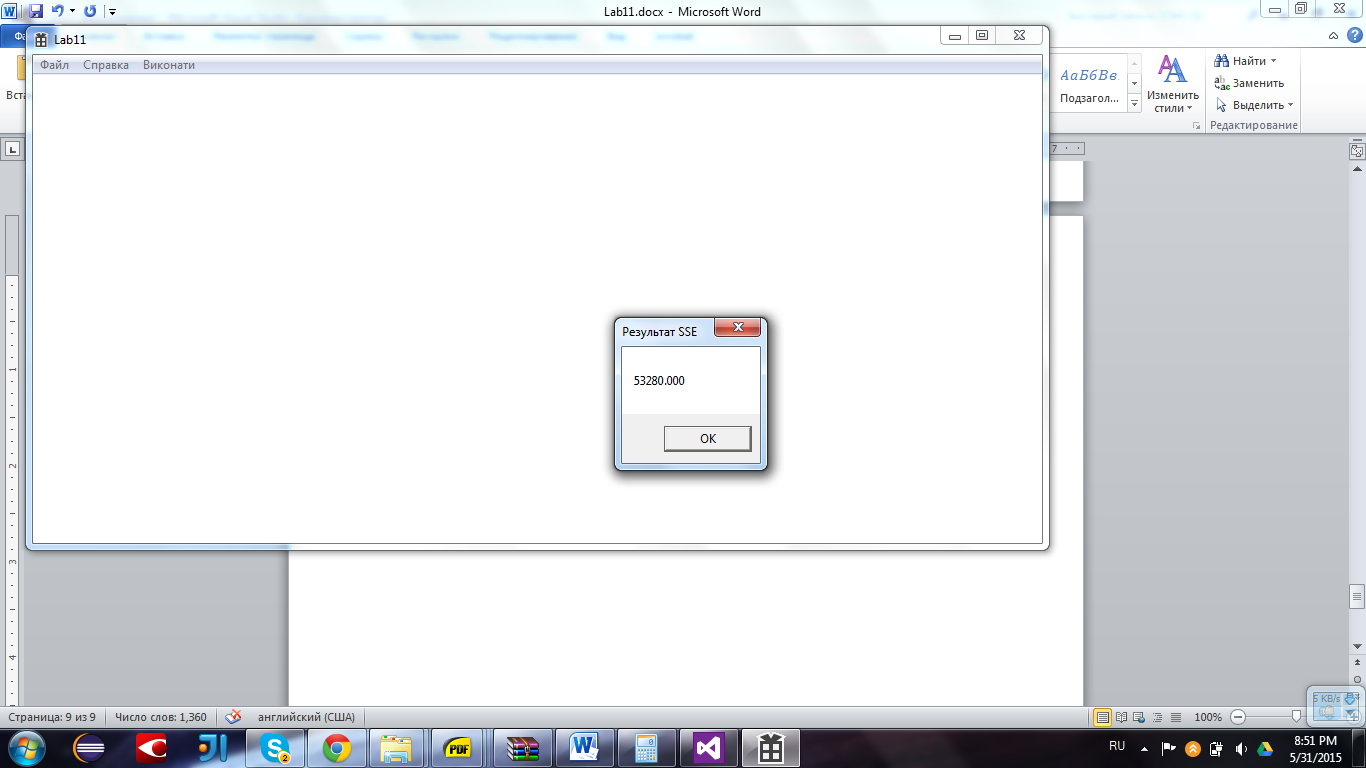
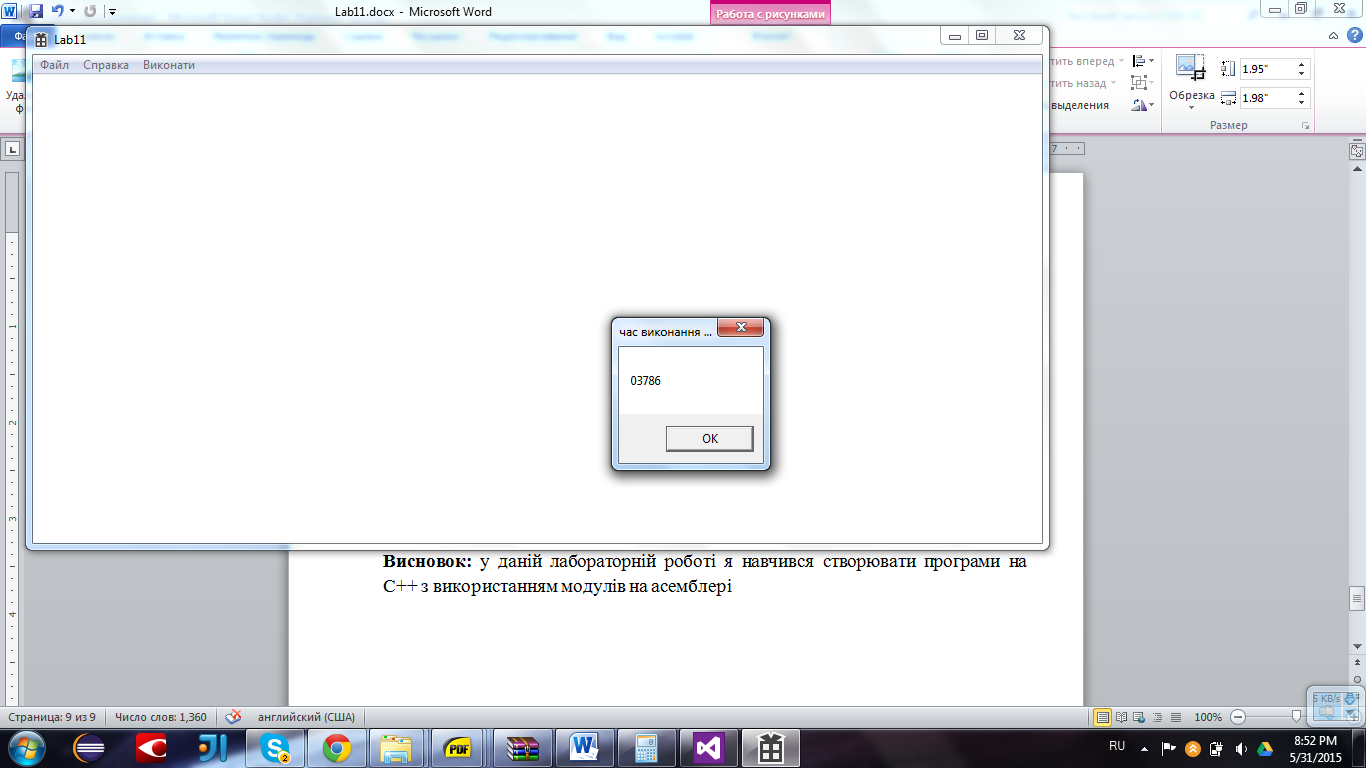
ret

vectorFPU endp

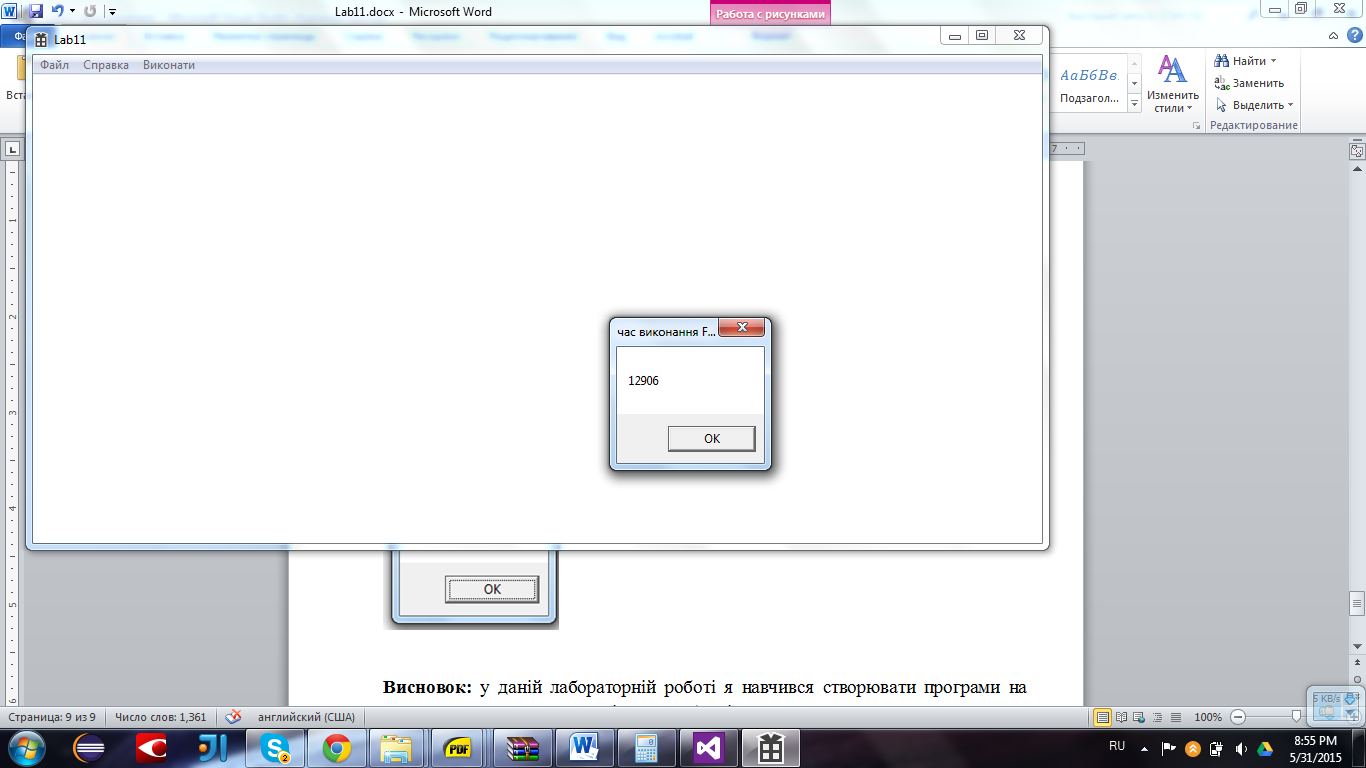
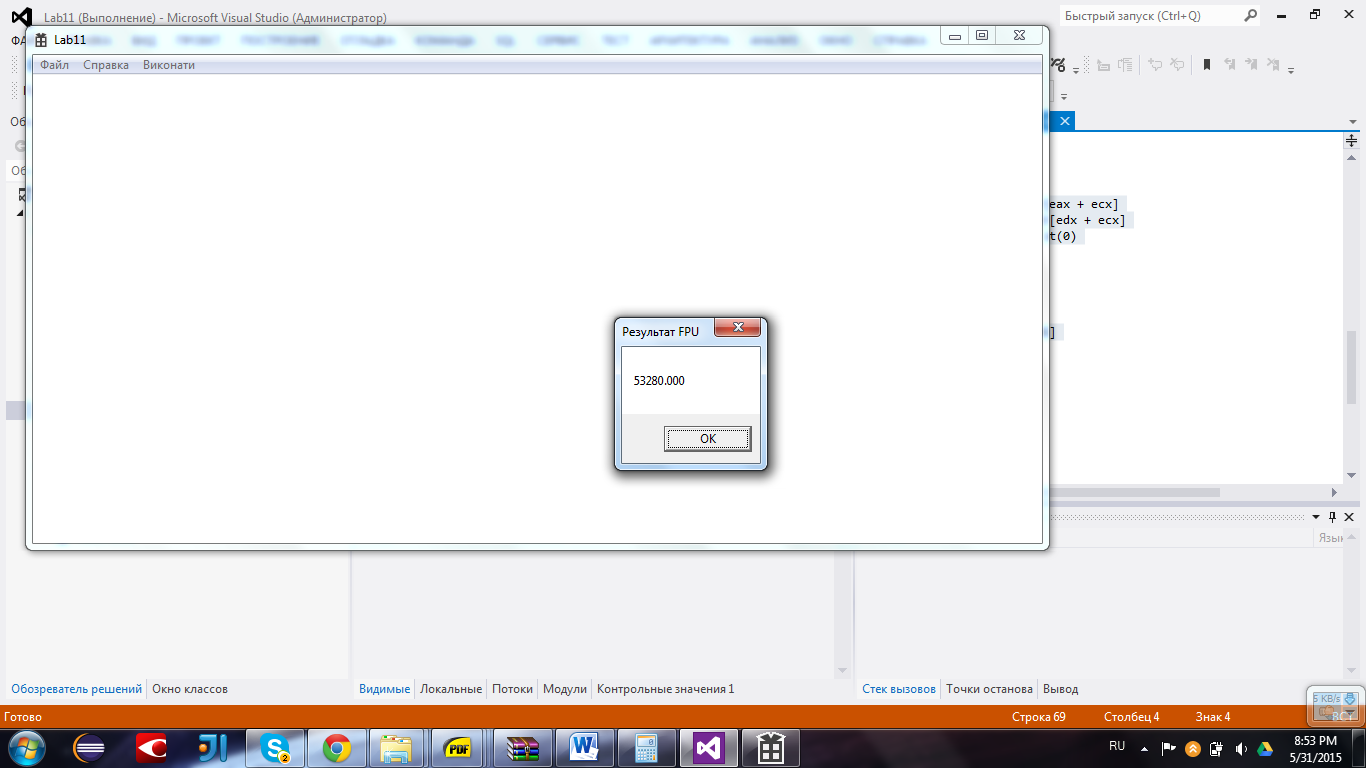
end

**Результати роботи програми**:

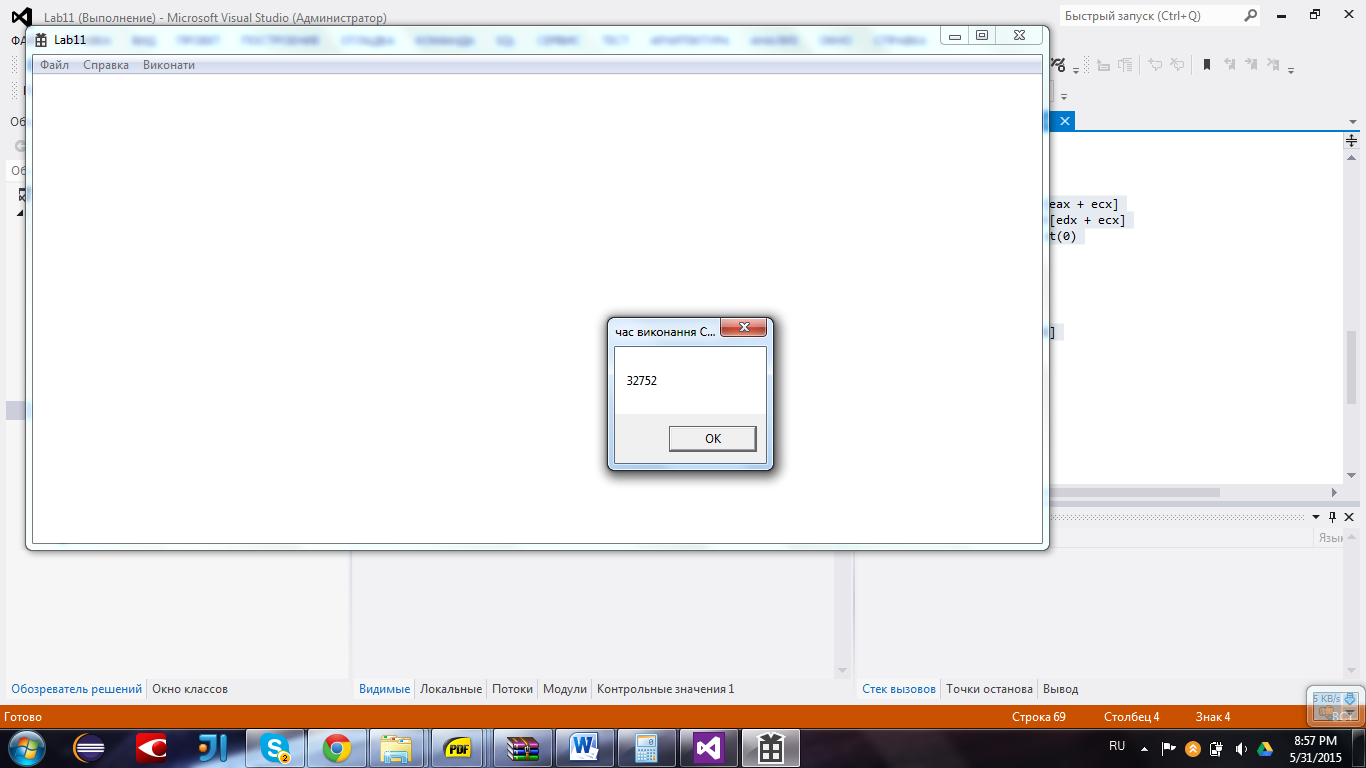
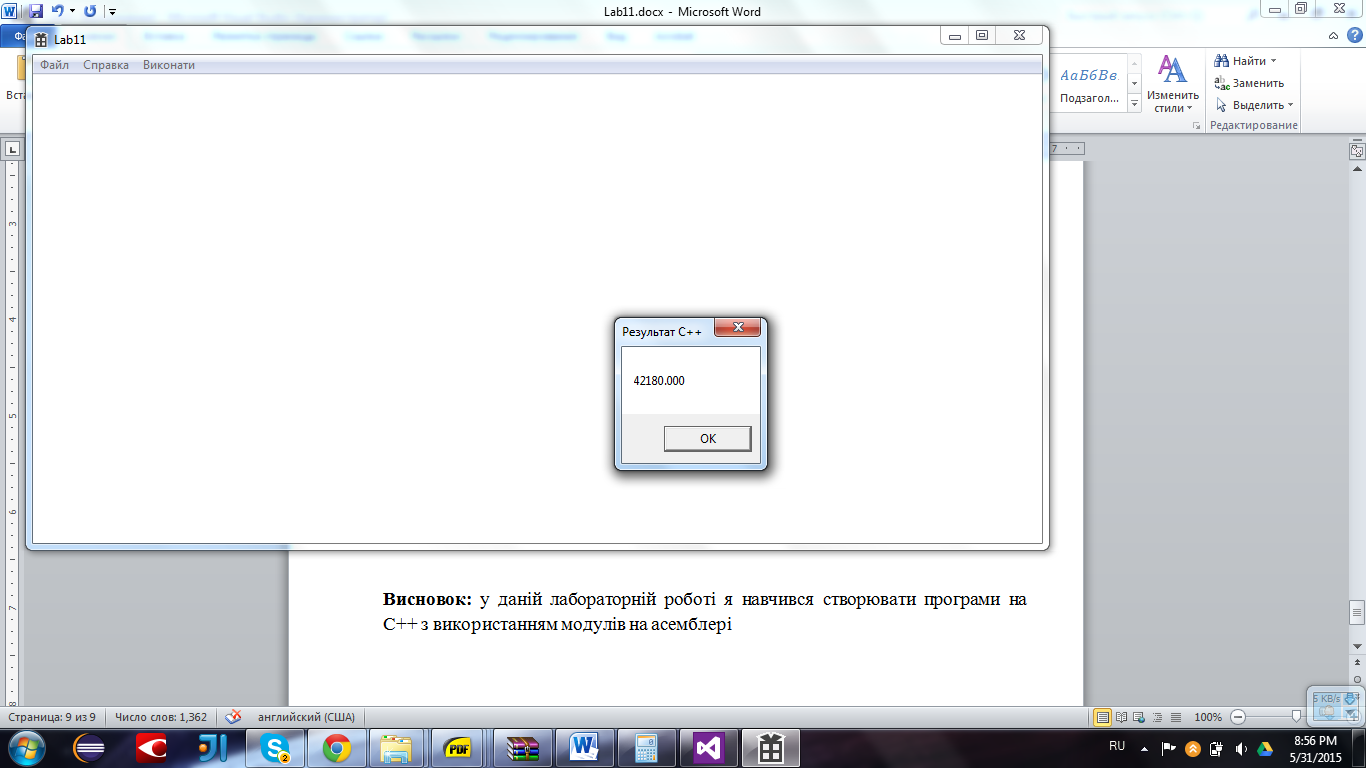
SSE

FPU



C++



**Висновок:** у даній лабораторній роботі я навчився програмувати модулі на асемблері, у яких містяться команди SSE, команди x87 FPU, а також використовувати такі модулі у проектах C++.