НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №9

з дісципліни **«**Системне програмування**»**

Виконав:

студент 2 курсу

ФІОТ гр. ІО-21

Кузьменко Володимир

Перевірив:

Порєв В.М.

Київ 2014 р.

**Мета:** Навчитися створювати програми на С++ з використанням модулів на

асемблері.

**Завдання:**

1. Створити у середовищі MS Visual Studio проект C++ з ім’ям Lab9.

2. Написати вихідний текст програми згідно варіанту завдання. У проекті

мають бути такі файли вихідного тексту:

- головний файл: lab9.cpp

- файли двох модулів на асемблері: module.asm та longop.asm.

3. У цьому проекті кожний модуль може окремо компілюватися.

4. Скомпілювати вихідний текст і отримати виконуємий файл програми.

5. Перевірити роботу програми. Налагодити програму.

6. Отримати результати – кодовані значення чисел згідно варіанту завдання.

7. Проаналізувати та прокоментувати результати, вихідний текст та машинний

код програми.

Текст програми:

Головний файл С++:

// lab9C.cpp: определяет точку входа для приложения.

//

#include "stdafx.h"

#include "lab9C.h"

#include "longop.h"

#include "module.h"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

TCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

TCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

void MyWork1(HWND hWnd);

int APIENTRY \_tWinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPTSTR lpCmdLine,

int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: разместите код здесь.

MSG msg;

HACCEL hAccelTable;

// Инициализация глобальных строк

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_LAB9C, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB9C));

// Цикл основного сообщения:

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// НАЗНАЧЕНИЕ: регистрирует класс окна.

//

// КОММЕНТАРИИ:

//

// Эта функция и ее использование необходимы только в случае, если нужно, чтобы данный код

// был совместим с системами Win32, не имеющими функции RegisterClassEx'

// которая была добавлена в Windows 95. Вызов этой функции важен для того,

// чтобы приложение получило "качественные" мелкие значки и установило связь

// с ними.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX 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\_LAB9C));

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_LAB9C);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassEx(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// НАЗНАЧЕНИЕ: сохраняет обработку экземпляра и создает главное окно.

//

// КОММЕНТАРИИ:

//

// В данной функции дескриптор экземпляра сохраняется в глобальной переменной, а также

// создается и выводится на экран главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance; // Сохранить дескриптор экземпляра в глобальной переменной

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// НАЗНАЧЕНИЕ: обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработка меню приложения

// WM\_PAINT -Закрасить главное окно

// WM\_DESTROY - ввести сообщение о выходе и вернуться.

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_COMMAND:

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case ID\_32772: // Вибір пункту меню

MyWork1(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:

hdc = BeginPaint(hWnd, &ps);

// TODO: добавьте любой код отрисовки...

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Обработчик сообщений для окна "О программе".

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 MyWork1(HWND hWnd)

{

long oA[12]={0x00000002,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000};

long oB[12]={0x00000001,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000};

long oN[12]={0x00000002,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000};

long oC[12]={0x00000004,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000};

long result1[384];

long result2[384];

long result3[384];

long result4[384];

long result5[24] ={0x00000000,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,

0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000};

char TextBuf[384];

Sub\_Long(result4, oB, oA);

Add\_Long(result2, oC, oN);

Multlong(result5, result4, result2);

StrHex\_MY(768, result5, TextBuf);

MessageBox(hWnd, TextBuf, "Результат A+B", MB\_OK);

}

Заголовочний файл до модулю logop.asm:

extern "C"

{

void Add\_Long(long \*dest, long \*pB, long \*pA);

void Sub\_Long(long \*dest, long \*pB, long \*pA);

void Factorial(long bits, long \*dest);

void Multlong (long \*dest, long \*pB, long \*pA);

void module\_changer(long n, long m, long \*src);

void module\_div10(long \*last, long \*dest, long n, long \*src);

void great();

}

Заголовочний файл до модулю module.asm:

extern "C"

{

void StrHex\_MY(long bits, long \*src, char \*dest);

void StrDec(long bits, long \*src, char \*dest);

}

Модуль longop.asm адаптований:

.586

.model flat, c

.data

level1 db 432 dup(0)

level2 db 432 dup(0)

counter db 40 dup(0)

variable db 40 dup(0)

cf db 1 dup(0)

cf2 db 1 dup(0)

temp db 1 dup(0)

regeax db 10 dup(0)

regebx db 10 dup(0)

regecx db 10 dup(0)

regedx db 10 dup(0)

.code

Add\_Long proc dest:DWORD, pB:DWORD, pA:DWORD

mov esi, pA ;[ebp+16] a

mov ebx, pB ;[ebp+12] b

mov edi, dest ;[ebp+8] res

mov edx, 0

mov ecx, 384

addAB:

mov eax, dword ptr[esi+edx]

adc eax, dword ptr[ebx+edx]

mov dword ptr [edi+edx], eax

inc edx

inc edx

inc edx

inc edx

dec ecx

dec ecx

dec ecx

dec ecx

jnz addAB

ret 12

Add\_Long endp

Sub\_Long proc dest:DWORD, pB:DWORD, pA:DWORD

mov esi, pA ;[ebp+16] ;a

mov ebx, pB ;[ebp+12] ;b

mov edi, dest ; [ebp+8] ;res

mov ecx, 384

mov edx, 0

subAB:

mov eax, dword ptr[esi+edx]

sbb eax, dword ptr[ebx+edx]

mov dword ptr [edi+edx], eax

inc edx

inc edx

inc edx

inc edx

dec ecx

dec ecx

dec ecx

dec ecx

jnz subAB

ret 12

Sub\_Long endp

Factorial proc

mov ebx, [ebp+12] ;адреса результату

mov edi, [ebp+8] ;факторіал числа

add dword ptr[ebx], 00000001h

mov ecx, edi

mov dword ptr[counter], 00000001h

factcycle:

mov dword ptr [variable], ecx

mov ecx, 56

mov esi, 0

level1cycle:

mov eax, dword ptr[ebx + esi]

mul dword ptr [counter]

mov dword ptr[level1 + esi], eax

mov dword ptr[level1 + 4 + esi], edx

add esi, 8

sub ecx, 8

jnz level1cycle

mov ecx, 48

mov esi, 0

level2cycle:

mov eax, dword ptr[ebx + 4 + esi]

mul dword ptr[counter]

mov dword ptr[level2 + 4 + esi], eax

mov dword ptr[level2 + 8 + esi], edx

add esi, 8

sub ecx, 8

jnz level2cycle

mov ecx, 56

mov esi, 0

sumlevelscycle:

mov eax, dword ptr[level1 + esi]

add cf, 0ffh

adc eax, dword ptr[level2 + esi]

mov cf, 0

adc cf, 0

mov [ebx + esi], eax

add esi, 4

sub ecx, 4

jnz sumlevelscycle

inc dword ptr [counter]

mov ecx, dword ptr [variable]

dec ecx

jnz factcycle

; mov ecx, dword ptr[counter]

; mov dword ptr[ebx], ecx

ret

Factorial endp

Multlong proc dest:DWORD, pB:DWORD, pA:DWORD

mov esi, pA ;[ebp+16] ; a

mov ebx, pB ;[ebp+12] ; b

mov edi, dest ;[ebp+8] ; res

clc

mov ecx, 28

mov dword ptr[counter], 0

level0cycle:

mov eax, dword ptr[ebx]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + ecx], eax

adc dword ptr[edi + 4 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level0cycle

clc

; mov cf, 00000000b

mov ecx, 28

mov dword ptr[counter], 0

level1cycle:

mov eax, dword ptr[ebx + 4]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 4 + ecx], eax

adc dword ptr[edi + 8 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level1cycle

clc

mov ecx, 28

mov dword ptr[counter], 0

level2cycle:

mov eax, dword ptr[ebx + 8]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 8 + ecx], eax

adc dword ptr[edi + 12 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level2cycle

clc

mov ecx, 28

mov dword ptr[counter], 0

level3cycle:

mov eax, dword ptr[ebx + 12]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 12 + ecx], eax

adc dword ptr[edi + 16 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level3cycle

clc

mov ecx, 28

mov dword ptr[counter], 0

level4cycle:

mov eax, dword ptr[ebx + 16]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 16 + ecx], eax

adc dword ptr[edi + 20 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level4cycle

clc

mov ecx, 28

mov dword ptr[counter], 0

level5cycle:

mov eax, dword ptr[ebx + 20]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 20 + ecx], eax

adc dword ptr[edi + 24 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level5cycle

clc

mov ecx, 28

mov dword ptr[counter], 0

level6cycle:

mov eax, dword ptr[ebx + 24]

mov dword ptr[variable], ecx

mov ecx, dword ptr[counter]

mul dword ptr[esi + ecx]

clc

add cf, 11111111b

adc dword ptr[edi + 24 + ecx], eax

adc dword ptr[edi + 28 + ecx], edx

mov cf, 00000000b

adc cf, 0

add dword ptr [counter], 4

mov ecx, dword ptr[variable]

sub ecx, 4

jnz level6cycle

ret 12

Multlong endp

module\_changer proc

mov edi, [ebp+16] ; sourse/res

mov ebx, [ebp+12] ; m

mov edx, [ebp+8] ; n

; not dword ptr [edi]

mov ecx, 544

mov esi, 0

cycle:

mov dword ptr[edi + esi], 0ffffffffh

add esi, 4

sub ecx, 4

jnz cycle

mov ecx, ebx

cmp ecx, 0

jz over

mov esi, edx

cycle2:

mov ebx, esi ; esi – це номер біту

mov edx, ebx

shr ebx, 3 ; номер байту

and edx, 07h ; позиція потрібного біту у байті

mov eax, 1

mov dword ptr [temp], ecx

mov ecx, edx

shl eax, cl ; маска 0..010..0 за умовчанням

mov ecx, dword ptr [temp]

not eax ; маска 1..101..1 для AND

and dword ptr [edi + ebx], eax

add esi, 1

sub ecx, 1

jnz cycle2

over:

ret 12

module\_changer endp

module\_div10 proc

mov esi, [ebp+20] ; sourse

mov edi, [ebp+16] ; n-bit

mov ebx, [ebp+12] ; res

mov edx, [ebp+8] ; last

mov eax, edi

shr eax, 3

mov dword ptr[counter], 0h

xor ecx, ecx

mov cl, byte ptr[esi + eax - 1]

shr cl, 4

sub edi, 4

again:

cmp cl, 10

jl little

sub cl, 10

mov dword ptr[regeax], eax

xor eax, eax

mov dword ptr[regecx], ecx

xor ecx, ecx

mov edx, edi

mov ecx, edx

shr edx, 3

and ecx, 07h

mov al, 1

shl al, cl

; not al

or byte ptr[ebx + edx], al

mov eax, dword ptr[regeax]

mov edx, [ebp+8]

mov ecx, dword ptr[regecx]

little:

dec edi

cmp edi, 0

jl exit

shl cl, 1

mov dword ptr[regeax], eax

xor eax, eax

mov dword ptr[regecx], ecx

xor ecx, ecx

mov edx, edi ; Nbit – це номер біту

mov ecx, edx

shr edx, 3 ; номер байту

and ecx, 07h ; бітова позиція = вирізаємо 3 молодші біти

mov al, 1

shl al, cl ; AL = маска вирізання біту Nbit

mov ah, byte ptr [esi + edx]

and ah, al ; результат у регістрі AH

mov edx, [ebp+8]

mov ecx, dword ptr[regecx]

cmp ah, 0

je here

add cl, 01h

here:

mov eax, dword ptr[regeax]

jmp again

exit:

mov byte ptr[edx], cl

ret 16

module\_div10 endp

great proc

ret

great endp

end

Модуль module.asm адаптований:

.586

.model flat, c

include \masm32\modules\longop.inc

.data

temp db 544 dup(0)

number db 544 dup(0)

last db 544 dup(0)

count db 10 dup(0)

result db 544 dup(0)

reggeax db 10 dup(0)

reggebx db 10 dup(0)

reggecx db 10 dup(0)

reggedx db 10 dup(0)

reggesi db 10 dup(0)

reggedi db 10 dup(0)

variable db 10 dup(0)

.code

;процедура StrHex\_MY записує текст шістнадцятькового коду

;перший параметр - адреса буфера результату (рядка символів)

;другий параметр - адреса числа

;третій параметр - розрядність числа у бітах (має бути кратна 8)

StrHex\_MY proc bits:DWORD, src:DWORD, dest:DWORD

mov ecx, bits ;[ebp+8] ;кількість бітів числа

cmp ecx, 0

jle @exitp

shr ecx, 3 ;кількість байтів числа

mov esi, src ; [ebp+12] ;адреса числа

mov ebx, dest ;[ebp+16] ;адреса буфера результату

@cycle:

mov dl, byte ptr[esi+ecx-1] ;байт числа - це дві hex-цифри

mov al, dl

shr al, 4 ;старша цифра

call HexSymbol\_MY

mov byte ptr[ebx], al

mov al, dl ;молодша цифра

call HexSymbol\_MY

mov byte ptr[ebx+1], al

mov eax, ecx

cmp eax, 4

jle @next

dec eax

and eax, 3 ;проміжок розділює групи по вісім цифр

cmp al, 0

jne @next

mov byte ptr[ebx+2], 32 ;код символа проміжку

inc ebx

@next:

add ebx, 2

dec ecx

jnz @cycle

mov byte ptr[ebx], 0 ;рядок закінчується нулем

@exitp:

ret

StrHex\_MY endp

;ця процедура обчислює код hex-цифри

;параметр - значення AL

;результат -> AL

HexSymbol\_MY proc

and al, 0Fh

add al, 48 ;так можна тільки для цифр 0-9

cmp al, 58

jl @exitp

add al, 7 ;для цифр A,B,C,D,E,F

@exitp:

ret

HexSymbol\_MY endp

StrDec proc

mov dword ptr[count], 0

mov eax, [ebp + 16] ; result

mov ebx, [ebp + 12] ; number

mov ecx, [ebp + 8] ; n-bit

cmp ecx, 0 ; якщо розрядність 0 - на вихід

je exit ;

xor edi, edi ;

cycle22: ;

mov dword ptr[result + edi], 0 ;

mov dword ptr[number + edi], 0 ;

add edi, 4 ;

cmp edi, 544 ;

jl cycle22

xor edi, edi ;

cycle1: ;

mov esi, dword ptr[ebx + edi] ; заносимо в масив діленого число для конвертування

mov dword ptr[number + edi], esi ;

add edi, 4 ;

cmp edi, 544 ;

je again ;

jmp cycle1

again:

;-----------------------------------------------------------------------------

;---------запис залишку в молодші розряди байта--------------------------------

;-----------------------------------------------------------------------------

mov dword ptr[reggecx], ecx ;

mov dword ptr[reggedx], edx ; зберігаємо стани регістрів

mov dword ptr[reggesi], esi ;

mov dword ptr[reggedi], edi ;

push offset number ;

push [ebp + 8] ;

push offset result ; викликаємо ділення

push offset last ;

call module\_div10 ;

; add esp, 16

mov ecx, dword ptr[last]

; mov edx, dword ptr[result]

mov ecx, dword ptr[reggecx] ;

mov edx, dword ptr[reggedx] ; відновлюємо стани регістрів

mov esi, dword ptr[reggesi] ;

mov edi, dword ptr[reggedi] ;

xor ecx, ecx ;

xor edi, edi ;

cycle2: ;

mov esi, dword ptr[result + edi] ;

cmp esi, 0 ;

je nothing ;

inc ecx ; перевіряємо результат ділення на 0

nothing: ;

add edi, 4 ;

cmp edi, 544 ;

je next1 ;

jmp cycle2 ;

next1:

mov eax, [ebp + 16] ;

mov dl, byte ptr[last] ; заносимо залишок від ділення в результат

mov esi, dword ptr[count] ;

add byte ptr[eax + esi], dl ;

cmp ecx, 0 ; якщо результат ділення 0 - на вихід

je exit ;

xor edi, edi ;

cycle: ;

mov esi, dword ptr[result + edi] ;

mov dword ptr[result + edi], 0h ; переносимо результат ділення в масив діленого

mov dword ptr[number + edi], esi ;

add edi, 4 ;

cmp edi, 544 ;

je nexth ;

jmp cycle ;

nexth:

;-----------------------------------------------------------------------------

;---------запис залишку в старші розряди байта--------------------------------

;-----------------------------------------------------------------------------

mov dword ptr[reggecx], ecx ;

mov dword ptr[reggedx], edx ; зберігаємо стани регістрів

mov dword ptr[reggesi], esi ;

mov dword ptr[reggedi], edi ;

push offset number ;

push [ebp + 8] ;

push offset result ; викликаємо ділення

push offset last ;

call module\_div10 ;

; add esp, 16

mov ecx, dword ptr[last]

; mov edx, dword ptr[result]

mov ecx, dword ptr[reggecx] ;

mov edx, dword ptr[reggedx] ; відновлюємо стани регістрів

mov esi, dword ptr[reggesi] ;

mov edi, dword ptr[reggedi] ;

xor ecx, ecx ;

xor edi, edi ;

cycle2h: ;

mov esi, dword ptr[result + edi] ;

cmp esi, 0 ;

je nothingh ;

inc ecx ; перевіряємо результат ділення на 0

nothingh: ;

add edi, 4 ;

cmp edi, 544 ;

je next1h ;

jmp cycle2h ;

next1h:

mov eax, [ebp + 16] ;

mov dl, byte ptr[last] ; заносимо залишок від ділення в результат

shl dl, 4 ;

mov esi, dword ptr[count] ;

add byte ptr[eax + esi], dl ;

cmp ecx, 0 якщо результат ділення 0 - на вихід

je exit ;

xor edi, edi ;

cycleh: ;

mov esi, dword ptr[result + edi] ;

mov dword ptr[result + edi], 0h ; переносимо результат ділення в масив діленого

mov dword ptr[number + edi], esi ;

add edi, 4 ;

cmp edi, 544 ;

je next ;

jmp cycleh ;

next: ; повторюємо процес з початку

add dword ptr[count], 01h ;

jmp again ;

exit:

; mov eax, [ebp + 16]

; mov ebx, dword ptr[last]

; mov dword ptr[eax], ebx

ret 12

StrDec endp

end

Висновок:

Під час виконання даної лабораторної роботи були закріплені навички використання процедур написаних на асемблері в програмах С++. Під час реалізації розрахунку виразів були виявлені фатальні дефекти в процедурах, але всі вони вдало були усунені.