Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**ЛАБОРАТОРНА РОБОТА № 4**

з дисципліни «Системне програмування-1» на тему

«Програмування арифметичних операцій підвищеної розрядності»

ВИКОНАЛА:

студентка ІІ курсу ФІОТ

групи ІО-64

Бровченко Анастасія

Залікова - 6403

ПЕРЕВІРИВ:

ст.вик. Порєв В. М.

Київ – 2018

**Мета:** Навчитися програмувати на асемблері основні арифметичні операції підвищеної розрядності, а також отримати перші навички програмування власних процедур у модульному проекті.

**Завдання:**

1. Створити у середовищі MS Visual Studio проект з ім’ям **Lab4**.

2. Написати вихідний текст програми згідно варіанту завдання. У проекті мають бути три модуля на асемблері:

- головний модуль: файл **main4.asm**. Цей модуль створити та написати заново, частково використавши текст модуля main3.asm попередньої роботи №3;

- другий модуль: використати **module** попередньої роботи №3;

- третій модуль: створити новий з ім'ям **longop**.

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 дизасембльований машинний код програми.

Числові значення у шістнадцятковому коді розрядності згідно варіанту. Числові значення тут надані так: від молодших ліворуч до старших праворуч розрядів групами по 32 біт. Записані явно перші 128 біт, далі згідно варіанту.

Операнди для додавання:

**A = 80010001h, 80020001h, 80030001h, 80040001h, ...**

**B = 80000001h, 80000001h, 80000001h, 80000001h, ...**

а також:

**A = (N 32-біт), (N+1 32-біт), (N+2 32-біт), (N+3 32-біт), ...**

**B = 00000001h, 00000001h, 00000001h, 00000001h, ...**

де N – номер варіанту

Операнди для віднімання:

**A = 0**

**B = (N 32-біт), (N+1 32-біт), (N+2 32-біт), (N+3 32-біт), ...**

де N – номер варіанту

**Роздруківка тексту програми:**

**module.asm**

.586

.model flat, c

.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

push ebp

mov ebp,esp

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

cmp ecx, 0

jle @exitp

shr ecx, 3 ;кількість байтів числа

mov esi, [ebp+12] ;адреса числа

mov ebx, [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:

pop ebp

ret 12

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

;ця процедура записує 8 символів HEX коду числа

;перший параметр - 32-бітове число

;другий параметр - адреса буфера тексту

DwordToStrHex proc

push ebp

mov ebp,esp

mov ebx,[ebp+8] ;другий параметр

mov edx,[ebp+12] ;перший параметр

xor eax,eax

mov edi,7

@next:

mov al,dl

and al,0Fh ;виділяємо одну шістнадцяткову цифру

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

cmp ax,58

jl @store

add ax,7 ;для цифр A,B,C,D,E,F

@store:

mov [ebx+edi],al

shr edx,4

dec edi

cmp edi,0

jge @next

pop ebp

ret 8

DwordToStrHex endp

end

**module.inc**

EXTERN StrHex\_MY : proc

EXTERN DwordToStrHex : proc

**longop.asm**

.586

.model flat, c

.code

Add\_160\_LONGOP proc

push ebp

mov ebp,esp

mov esi, [ebp+16] ; ESI = адреса A

mov ebx, [ebp+12] ; EBX = адреса B

mov edi, [ebp+8] ; EDI = адреса результату

mov ecx, 5 ; ECX = потрібна кількість повторень

mov edx, 0

clc ; обнулює біт CF регістру EFLAGS

cycle:

mov eax, dword ptr[esi+4\*edx]

adc eax, dword ptr[ebx+4\*edx] ; додавання групи з 32 бітів

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

inc edx

dec ecx ; лічильник зменшуємо на 1

jnz cycle

pop ebp

ret 12

Add\_160\_LONGOP endp

Sub\_960\_LONGOP proc

push ebp

mov ebp,esp

mov esi, [ebp+16] ; ESI = адреса A

mov ebx, [ebp+12] ; EBX = адреса B

mov edi, [ebp+8] ; EDI = адреса результату

mov ecx, 30 ; ECX = потрібна кількість повторень

mov edx, 0

clc ; обнулює біт CF регістру EFLAGS

cycle:

mov eax, dword ptr[esi+4\*edx]

sbb eax, dword ptr[ebx+4\*edx] ; віднімання групи з 32 бітів

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

inc edx

dec ecx ; лічильник зменшуємо на 1

jnz cycle

pop ebp

ret 12

Sub\_960\_LONGOP endp

End

**longop.inc**

EXTERN Add\_160\_LONGOP : proc

EXTERN Sub\_960\_LONGOP : proc

**main4.asm**

.586

.model flat, stdcall

option casemap :none

include D:\masm32\include\kernel32.inc

include D:\masm32\include\user32.inc

include D:\masm32\include\windows.inc

include module.inc

include longop.inc

includelib D:\masm32\lib\kernel32.lib

includelib D:\masm32\lib\user32.lib

.const

.data

.data

Caption1 db "A+B 1",0

Caption3 db "A+B 2",0

Caption2 db "A-B",0

TextBuf1 db 160 dup(?)

TextBuf3 db 160 dup(?)

TextBuf2 db 960 dup(?)

ValueA1 dd 160 dup(?)

ValueB1 dd 160 dup(?)

ValueA3 dd 160 dup(?)

ValueB3 dd 160 dup(?)

ValueA2 dd 960 dup(?)

ValueB2 dd 960 dup(?)

Result1 dd 160 dup(0)

Result3 dd 160 dup(0)

Result2 dd 960 dup(0)

.code

main:

;A+B 1

mov eax, 80010001h

mov ecx, 5 ; ECX = потрібна кількість повторень

mov edx, 0

cycleAB1:

mov DWord ptr[ValueA1+4\*edx], eax

mov DWord ptr[ValueB1+4\*edx], 80000001h

add eax, 10000h

inc edx

dec ecx ; зменшуємо лічильник на 1

jnz cycleAB1

push offset ValueA1

push offset ValueB1

push offset Result1

call Add\_160\_LONGOP

push offset TextBuf1

push offset Result1

push 160

call StrHex\_MY

invoke MessageBoxA, 0, ADDR TextBuf1, ADDR Caption1,0

;A+B 2

mov eax, 3h

mov ecx, 5 ; ECX = потрібна кількість повторень

mov edx,0

cycleAB3:

mov DWord ptr[ValueA3+4\*edx], eax

mov DWord ptr[ValueB3+4\*edx], 00000001h

add eax, 1h

inc edx

dec ecx ; зменшуємо лічильник на 1

jnz cycleAB3

push offset ValueA3

push offset ValueB3

push offset Result3

call Add\_160\_LONGOP

push offset TextBuf3

push offset Result3

push 160

call StrHex\_MY

invoke MessageBoxA, 0, ADDR TextBuf3, ADDR Caption3,0

;A-B

mov eax, 3h

mov ecx, 30 ; ECX = потрібна кількість повторень

mov edx,0

cycleAB2:

mov DWord ptr[ValueA2+4\*edx], 0

mov DWord ptr[ValueB2+4\*edx], eax

add eax, 1h

inc edx

dec ecx ; зменшуємо лічильник на 1

jnz cycleAB2

push offset ValueA2

push offset ValueB2

push offset Result2

call Sub\_960\_LONGOP

push offset TextBuf2

push offset Result2

push 960

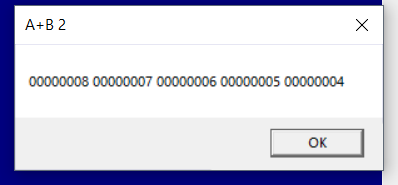
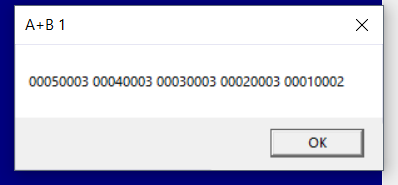
call StrHex\_MY

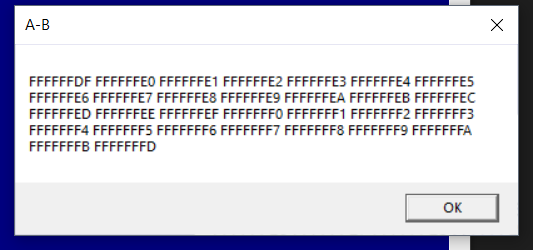
invoke MessageBoxA, 0, ADDR TextBuf2, ADDR Caption2,0

invoke ExitProcess, 0

end main

**Результати роботи програми:**



****

**Аналіз результатів:**

Дана програма виконує операції віднімання і додавання з числами підвищеної розрядності.

Операції ADC та SBB дозволяють нам додавати та віднімати з переносом (ADC) чи позичанням біту (SBB).Наприклад, якщо деяка група бітів містить 32 біти, то результат додавання буде 33 –бітним. 33 біт – біт переносу – автоматично записується у біт CF регістру EFLAGS.

**Висновок:**

Під час виконання лабораторної роботи були покращені навички написання власних модулів, роботи з циклами, а також були закріпленні основні навички в операціях додавання і віднімання чисел з підвищеною розрядністю.