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

Системне програмування

Лабораторна робота №3

«Програмування арифметичних операцій
підвищеної розрядності»

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

Перевірив:
Порєв В. М.

Київ - 2024

Тема: Програмування арифметичних операцій підвищеної розрядності.

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

Завдання:

1. Створити у середовищі Microsoft Visual Studio проект з ім’ям **Lab3**.

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

- головний модуль: файл **main3.asm**. Цей модуль створити та написати заново, частково використавши текст модуля main2.asm попередньої роботи №2;

- другий модуль: використати **module** попередньої роботи №2;

- 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 дизасемблерний машинний код програми.

Індивідуальний варіант завдання:

Номер варіанту визначається згідно списку студентів у журналі.

Необхідно запрограмувати два додавання (A + B) та два віднімання (A - B).

Згідно мого варіанту, операнди A та B є наступними, перше додавання:

A = 80010001h, 80020001h, 80030001h, 80040001h, 80050001h, 80060001h, 80070001h, 80080001h, 80090001h, 800A0001h, 800B0001h, 800C0001h, 800D0001h, 800E0001h, 800F0001h, 80100001h.

B = 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h.

Друге додавання:

A = 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h, 00000013h, 00000014h, 00000015h, 00000016h, 00000017h, 00000018h, 00000019h, 0000001Ah, 0000001Bh, 0000001Ch, 0000001Dh.

B = 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h, 80000003h.

Перше віднімання:

A = 00000000h, 00000000h.

B = 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h, 00000013h, 00000014h, 00000015h, 00000016h, 00000017h, 00000018h, 00000019h, 0000001Ah, 0000001Bh, 0000001Ch, 0000001Dh, 0000001Eh, 0000001Fh, 00000020h.

Друге віднімання:

A = 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h, 00000013h, 00000014h, 00000015h, 00000016h, 00000017h, 00000018h, 00000019h, 0000001Ah, 0000001Bh, 0000001Ch, 0000001Dh, 0000001Eh, 0000001Fh, 00000020h.

B = 00000001h, 00000001h.

Розрядність додавання (біт): 512.

Розрядність віднімання (біт): 608.

Виконання завдання:

Роздруківка коду програми:

module.inc:

```
EXTERN StrHex_MY : proc
```

longop.inc:

```
EXTERN Add_512_LONGOP : proc
EXTERN Sub_608_LONGOP : proc
```

longop.asm:

```
.586
.model flat, c

.code
Add_512_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, 15                ;у регістр ECX записуємо кількість повторень
      mov edx, 1                       ;у регістр EDX записуємо позицію
зсуву
      clc                        ;обнулює біт CF регістру EFLAGS

      mov eax, dword ptr[esi]    ;починаємо з молодших груп
      add eax, dword ptr[ebx]    ;додавання 32-біт
      mov dword ptr[edi], eax    ;запис молодших 32 біт суми

      cycle:
            mov eax, dword ptr[esi+edx*4] ;наступна 32-бітова група
            adc eax, dword ptr[ebx+edx*4] ;додавання з урахуванням переносу
з попередньої групи
            mov dword ptr[edi+edx*4], eax ;запис наступних 32 біт суми
            inc edx                                          ;збільшуємо
позицію зсуву на 1
            dec ecx                                          ;зменшуємо
лічильник на 1
            jnz cycle                                  ;якщо лічильник не 0,
то перехід на мітку cycle

      mov eax, 0
      adc eax, 0
      mov dword ptr [edi+16*4], eax
```

```
      pop ebp                   ;відновлення стеку
      ret 12

Add_512_LONGOP endp

Sub_608_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, 18                     ;у регістр ECX записуємо кількість
повторень
      mov edx, 1                      ;у регістр EDX записуємо позицію
зсуву
      clc                       ;обнулює біт CF регістру EFLAGS

      mov eax, dword ptr[esi]   ;починаємо з молодших груп
      sub eax, dword ptr[ebx]   ;віднімання 32-біт
      mov dword ptr[edi], eax   ;запис молодших 32 біт різниці

      cycle:
            mov eax, dword ptr[esi+edx*4] ;наступна 32-бітова група
            sbb eax, dword ptr[ebx+edx*4] ;віднімання з урахуванням
переносу з попередньої групи
            mov dword ptr[edi+edx*4], eax ;запис наступних 32 біт різниці
            inc edx                                         ;збільшуємо
позицію зсуву на 1
            dec ecx                                         ;зменшуємо
лічильник на 1
            jnz cycle                                 ;якщо лічильник не 0,
то перехід на мітку cycle

      pop ebp                   ;відновлення стеку
      ret 12

Sub_608_LONGOP endp

end
```

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

end
```

main3.asm:

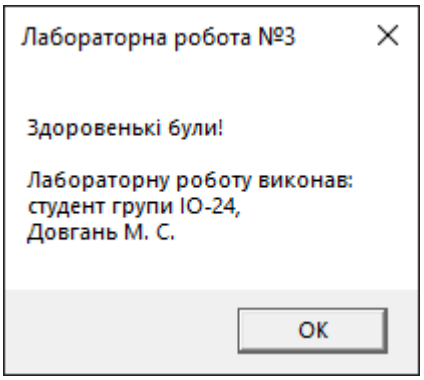
```
.586
.model flat, stdcall

include module.inc
include longop.inc

include \masm32\include\user32.inc
include \masm32\include\kernel32.inc


.data
      mainWindowTitle db "Лабораторна робота №3", 0
      mainWindowText db "Здоровенькі були!", 13, 10, 13, 10,
                              "Лабораторну роботу виконав:", 13, 10,
                              "студент групи ІО-24,", 13, 10,
                              "Довгань М. С.", 0

      firstAddA dd 16 dup(?)
      titleAddFirstA db "Значення A", 0

      firstAddB dd 16 dup(?)
      titleAddFirstB db "Значення B", 0

      firstAddRes dd 17 dup(0)
      titleAddFirstRes db "Перше додавання", 0

      secondAddA dd 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h,
00000013h, 00000014h, 00000015h,
                           00000016h, 00000017h, 00000018h, 00000019h,
0000001Ah, 0000001Bh, 0000001Ch, 0000001Dh
      secondAddB dd 80000003h, 80000003h, 80000003h, 80000003h, 80000003h,
80000003h, 80000003h, 80000003h,
                           80000003h, 80000003h, 80000003h, 80000003h,
80000003h, 80000003h, 80000003h, 80000003h
```

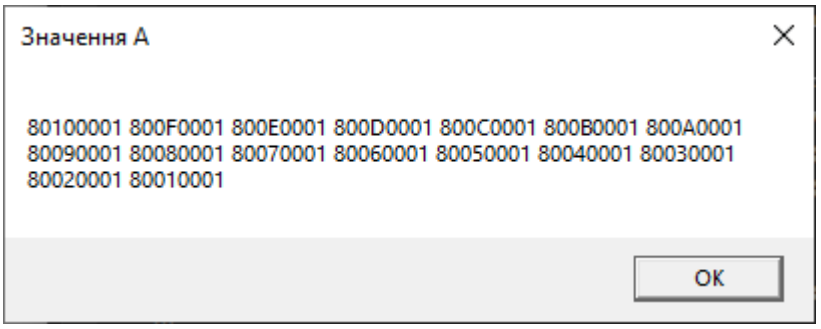
```
	secondAddRes dd 17 dup(0)
	titleAddSecondRes db "Друге додавання", 0

	firstSubA dd 00000000h, 00000000h, 00000000h, 00000000h, 00000000h,
00000000h, 00000000h,
			00000000h, 00000000h, 00000000h, 00000000h,
00000000h, 00000000h, 00000000h,
			00000000h, 00000000h, 00000000h, 00000000h,
00000000h
	firstSubB dd 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h,
00000013h, 00000014h,
			00000015h, 00000016h, 00000017h, 00000018h,
00000019h, 0000001Ah, 0000001Bh,
			0000001Ch, 0000001Dh, 0000001Eh, 0000001Fh,
00000020h

	firstSubRes dd 19 dup(0)
	titleSubFirstRes db "Перше віднімання", 0

	secondSubA dd 0000000Eh, 0000000Fh, 00000010h, 00000011h, 00000012h,
00000013h, 00000014h,
			00000015h, 00000016h, 00000017h, 00000018h,
00000019h, 0000001Ah, 0000001Bh,
			0000001Ch, 0000001Dh, 0000001Eh, 0000001Fh,
00000020h
	secondSubB dd 00000001h, 00000001h, 00000001h, 00000001h, 00000001h,
00000001h, 00000001h,
			00000001h, 00000001h, 00000001h, 00000001h,
00000001h, 00000001h, 00000001h,
			00000001h, 00000001h, 00000001h, 00000001h,
00000001h

	secondSubRes dd 19 dup(0)
	titleSubSecondRes db "Друге віднімання", 0

	lastWindowTitle db "Програма завершила роботу", 0
	lastWindowText db "Дякую за увагу!", 0

	TextBuf dd 19 dup(0)


.code
main3:
	invoke MessageBoxA, 0, ADDR mainWindowText, ADDR mainWindowTitle, 0

	mov eax, 16
	mov ebx, 0
	mov ecx, 80010001h
	cycleFirstAdd:
		mov dword ptr[firstAddA+4*ebx], ecx
```

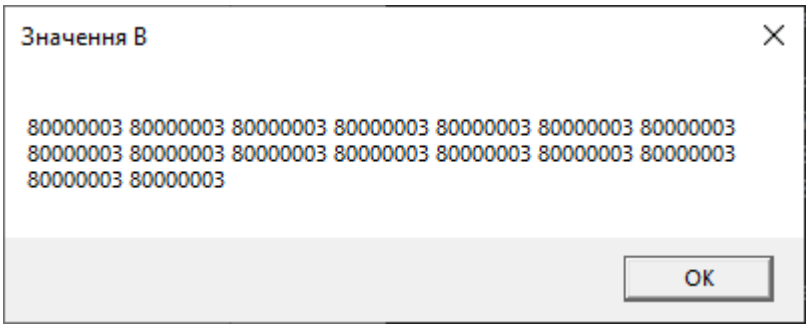
```
      mov dword ptr[firstAddB+4*ebx], 80000003h
      inc ebx
      add ecx, 10000h
      dec eax
      jnz cycleFirstAdd

push offset textbuf
push offset firstAddA
push 512
call StrHex_MY

invoke MessageBoxA, 0, ADDR textbuf, ADDR titleAddFirstA, 0

push offset textbuf
push offset firstAddB
push 512
call StrHex_MY

invoke MessageBoxA, 0, ADDR textbuf, ADDR titleAddFirstB, 0

;---------- First Add ----------

push offset firstAddA
push offset firstAddB
push offset firstAddRes
call Add_512_LONGOP

push offset textbuf
push offset firstAddRes
push 544
call StrHex_MY

invoke MessageBoxA, 0, ADDR textbuf, ADDR titleAddFirstRes, 0

;---------- Second Add ----------

push offset secondAddA
push offset secondAddB
push offset secondAddRes
call Add_512_LONGOP

push offset textbuf
push offset secondAddRes
push 544
call StrHex_MY

invoke MessageBoxA, 0, ADDR textbuf, ADDR titleAddSecondRes, 0

;---------- First Substract ----------
```

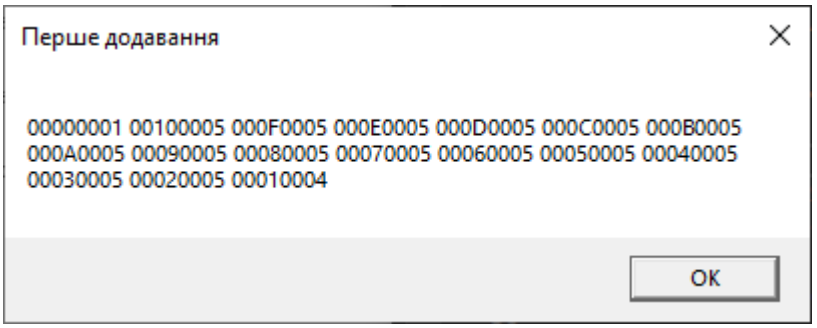
```
        push offset firstSubA
        push offset firstSubB
        push offset firstSubRes
        call Sub_608_LONGOP

        push offset textbuf
        push offset firstSubRes
        push 608
        call StrHex_MY

        invoke MessageBoxA, 0, ADDR textbuf, ADDR titleSubFirstRes, 0

        ;---------- Second Substract ----------

        push offset secondSubA
        push offset secondSubB
        push offset secondSubRes
        call Sub_608_LONGOP

        push offset textbuf
        push offset secondSubRes
        push 608
        call StrHex_MY

        invoke MessageBoxA, 0, ADDR textbuf, ADDR titleSubSecondRes, 0

        invoke MessageBoxA, 0, ADDR lastWindowText, ADDR lastWindowTitle, 0

        invoke ExitProcess, 0

end main3
```

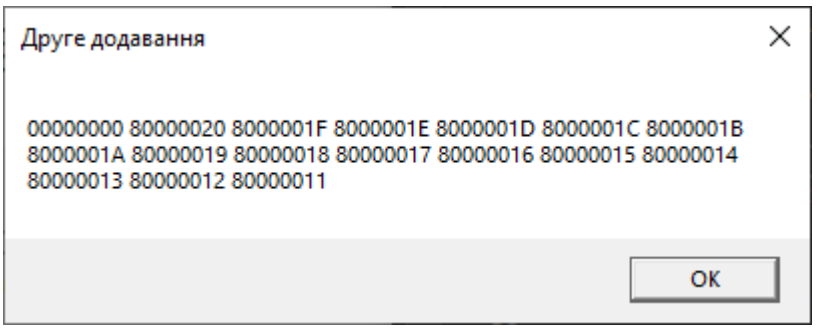
Результати виконання програми:

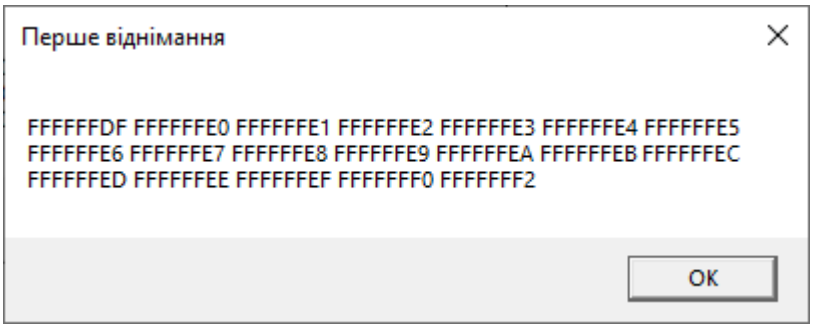


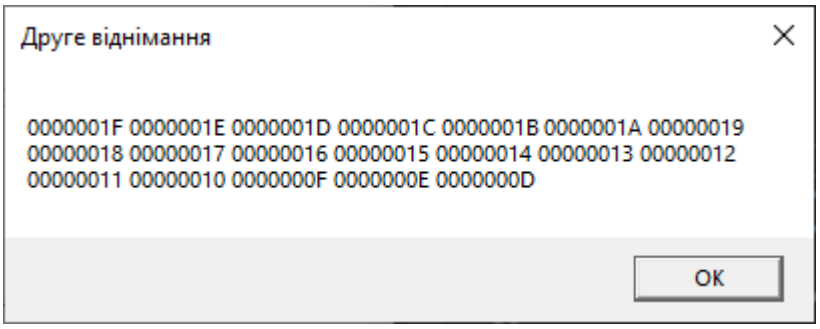


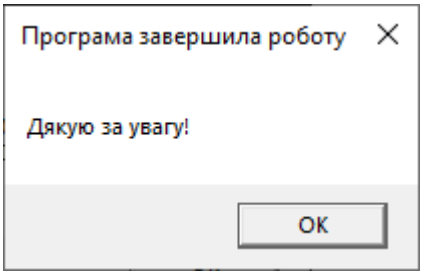





Друге додавання
00000000 80000020 8000001F 8000001E 8000001D 8000001C 8000001B
8000001A 80000019 80000018 80000017 80000016 80000015 80000014
80000013 80000012 80000011
OK


Перше віднімання
FFFFFFDF FFFFFFE0 FFFFFFE1 FFFFFFE2 FFFFFFE3 FFFFFFE4 FFFFFFE5
FFFFFFE6 FFFFFFE7 FFFFFFE8 FFFFFFE9 FFFFFFEA FFFFFFEB FFFFFFEC
FFFFFFED FFFFFFEE FFFFFFEF FFFFFFF0 FFFFFFF2
OK


Друге віднімання
0000001F 0000001E 0000001D 0000001C 0000001B 0000001A 00000019
00000018 00000017 00000016 00000015 00000014 00000013 00000012
00000011 00000010 0000000F 0000000E 0000000D
OK


Програма завершила роботу
Дякую за увагу!
OK

Аналіз виконання програми:
Створена мною програма виконує завдання лабораторної роботи, відповідно до мого індивідуального варіанту завдання та обчислених значень. Усі ці обчислені значення заносяться до програми у форматі dd (define double word), тобто чотири байти. На початку виконання, програма видає користувачеві стартове вікно-привітання. Після цього в нас виводиться перше вікно - значення A, та за ним одразу друге - значення B. Потім програма виконує перше додавання згідно завдання, тобто (A + B), потім друге додавання. Після цього в нас відбувається процес першого віднімання та другого, із відповідними виведеннями вікон для користувача. Коли користувач уже ознайомився із усіма цими виводами, в нього з’являється останнє вікно, яке повідомляє його, що всі значення були виведені, та програма завершує свою роботу.

Висновок: під час виконання даної лабораторної роботи я навчився програмувати на Асемблері основні арифметичні операції підвищеної розрядності, а також отримав перші навички у програмуванні власних процедур у модульному проекті.