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 № 6**

з дисципліни «Системне програмування-1» на тему

«Програмування побітових операцій»

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 побітові операції, вивчити основні команди обробки бітів.

**Завдання:**

1. Створити у середовищі MS Visual Studio проект з ім’ям **Lab6**.

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

- головний модуль: файл **main6.asm**. Цей модуль створити та написати заново;

- другий модуль: використати **module** попередніх робіт;

- третій модуль: модуль **longop** попередньої роботи №5 доповнити новим кодом відповідно завданню.

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.

**Варіант завдання:**

Запис 1 у M бітів, починаючи з N-го розряду. Розрядність 1024.

**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\_864\_LONGOP proc

push ebp

mov ebp,esp

mov esi, [ebp+16]

mov ebx, [ebp+12]

mov edi, [ebp+8]

mov ecx, 0

addAB:

mov eax, dword ptr[esi+ecx]

adc eax, dword ptr[ebx+ecx]

mov dword ptr [edi+ecx], eax

add ecx, 4

cmp ecx, 864

jl addAB

pop ebp

ret 12

Add\_864\_LONGOP endp

Sub\_256\_LONGOP proc

push ebp

mov ebp,esp

mov esi, [ebp+16]

mov ebx, [ebp+12]

mov edi, [ebp+8]

mov ecx, 0

subAB:

mov eax, dword ptr[esi+ecx]

sbb eax, dword ptr[ebx+ecx]

mov dword ptr [edi+ecx], eax

add ecx, 4

cmp ecx, 256

jl subAB

pop ebp

ret 12

Sub\_256\_LONGOP endp

Mul\_N32\_LONGOP proc

push ebp

mov ebp, esp

mov esi, [ebp+16]

mov ebx, [ebp+12]

mov edi, [ebp+8]

xor ecx, ecx

@cycle:

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

mul ebx

add dword ptr[edi+4\*ecx], eax

add dword ptr[edi+4\*ecx+4], edx

inc ecx

cmp ecx, 7

jb @cycle

pop ebp

ret 12

Mul\_N32\_LONGOP endp

Mul\_N\_x\_N\_LONGOP proc

push ebp

mov ebp, esp

mov esi, [ebp + 16]

mov edi, [ebp + 12]

mov bl, [ebp + 8]

x db 0

mov x, bl

mov ecx, 8

xor ebx, ebx

@cycle1:

mov eax, dword ptr[edi + 8 \* ebx]

mul x

mov dword ptr[esi + 8 \* ebx], eax

mov dword ptr[esi + 8 \* ebx + 4], edx

inc ebx

dec ecx

jnz @cycle1

mov ecx, 8

xor ebx, ebx

@cycle2:

mov eax, dword ptr[edi + 8 \* ebx + 4]

mul x

clc

adc eax, dword ptr[esi + 8 \* ebx + 4]

mov dword ptr[esi + 8 \* ebx + 4], eax

clc

adc edx, dword ptr[esi + 8 \* ebx + 8]

mov dword ptr[esi + 8 \* ebx + 8], edx

inc ebx

dec ecx

jnz @cycle2

pop ebp

ret 12

Mul\_N\_x\_N\_LONGOP endp

Write\_1\_LONGOP proc

push ebp

mov ebp, esp

mov eax, [ebp+8]; m

mov edx, [ebp+12]; n

mov esi, [ebp+16]; adres

mov ebx, edx; n

and ebx, 7; 0000 0xxx

shr edx, 3; byte

mov ecx, ebx

mov ch, 255; 1111 1111

shl ch, cl; 1110 0000

xor ebx, 7; 5:101 = 010:2

inc ebx; 2 + 1 = 3

sub eax, ebx

jc @inner

or byte ptr[esi+edx], ch

inc edx

xor ecx, ecx

@byte1:

sub eax, 8

jc @out

or byte ptr[esi+edx], 255

inc edx

jmp @byte1

@inner:

not eax

inc eax

xor ebx, ebx

mov bh, ch

mov ecx, eax

shl bh, cl

jmp @end

@out:

not eax

inc eax

and eax, 7

xor ebx, ebx

mov bh, 255

mov ecx, eax

@end:

shr bh, cl

or byte ptr[esi+edx], bh

pop ebp

ret 12

Write\_1\_LONGOP endp

End

**longop.inc**

EXTERN Add\_864\_LONGOP : proc

EXTERN Sub\_256\_LONGOP : proc

EXTERN Mul\_N\_x\_N\_LONGOP : proc

EXTERN Mul\_N32\_LONGOP : proc

EXTERN Write\_1\_LONGOP : proc

**main.asm**

.586

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include module.inc

include longop.inc

include \masm32\include\kernel32.inc

include \masm32\include\user32.inc

includelib \masm32\lib\kernel32.lib

includelib \masm32\lib\user32.lib

.data

mas db 128 dup(0)

n dd 9

m dd 50

TextBuff1 db 256 dup(?)

TextBuff2 db 302 dup(?)

Caption1 db "Початкові дані" , 0

Caption2 db "Вихідні дані", 0

.code

main:

push offset TextBuff1

push offset mas

push 1024

call StrHex\_MY

invoke MessageBox, 0, ADDR TextBuff1, ADDR Caption1, MB\_ICONINFORMATION

push offset mas

push n

push m

call Write\_1\_LONGOP

push offset TextBuff2

push offset mas

push 1024

call StrHex\_MY

invoke MessageBox, 0, ADDR TextBuff2, ADDR Caption2, MB\_ICONINFORMATION

invoke ExitProcess, 0

end main

**Висновок:**

Під час виконання лабораторної роботи були покращені навички написання власних модулів, роботи з циклами. Наввчились програмувати побітові операції на асемблері, вивчили основні команди обробки бітів.