**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**Лабораторна робота №6**

з дисципліни

«Системне програмування»

на тему

«Обчислення арифметичних виразів і трансцендентних функцій.

Використання команд співпроцесора ix87»

Виконав:

Перевірив:

студент групи ІП-93

Павлов Валерій Георгійович

Домінський Валентин Олексійович

номер залікової книжки: 9311

номер у списку: 9

Київ 2021

**Мета:**

Вивчення команд Асемблера для арифметики з плаваючою комою і здобуття навичок виконання розрахунків з елементами масивів

**Порядок виконання роботи:**

1. Вивчити арифметичні команди з плаваючою комою
2. Розробити програму на мові Асемблер, в якій згідно з індивідуальним варіантом завдання виконуються обчислення значення арифметичного вираження із застосуванням команд співпроцесора ix87 з подальшим виведенням результату\* у віконному інтерфейсі

* Вхідні дані та результат у вікні виводу мають бути представлені у десяткової системі счислення

1. Для всіх варіантів передбачити завдання значень вхідних змінних у форматі double (DQ), проміжних результатів обчислень – у форматі long double (DT), а кінцевих - знову у форматі double
2. Розрахунки (п. 2, 3) повторити в програмі для 5 значень змінних\*\*, причому всі вхідні значення задати дійсними числами у вигляді одновимірних масивів

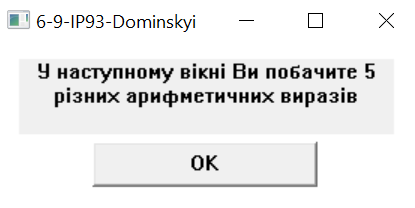
* Значення змінних a, b, c та d повинні мати бути дійсними числами, не рівними 0 або 1, допустимими для обчислення трансцендентних функцій, а тригонометричні функції обчислюються від аргументів, заданих в радіанах

1. Для перевірки правильності виконання розрахунків і результатів, що виводяться, заздалегідь виконати контрольні розрахунки. Проміжні і остаточні результати контрольних розрахунків з точністю не гірше точності розрахунків у комп'ютеру привести в звіті
2. Виконати відладку програми шляхом порівняння розрахованих програмою результатів з контрольними прикладами. Лістинг розробленої програми і скріншоти розрахунків по всіх контрольних прикладах привести в звіті по лабораторній роботі
3. Зробити висновки по лабораторній роботі.

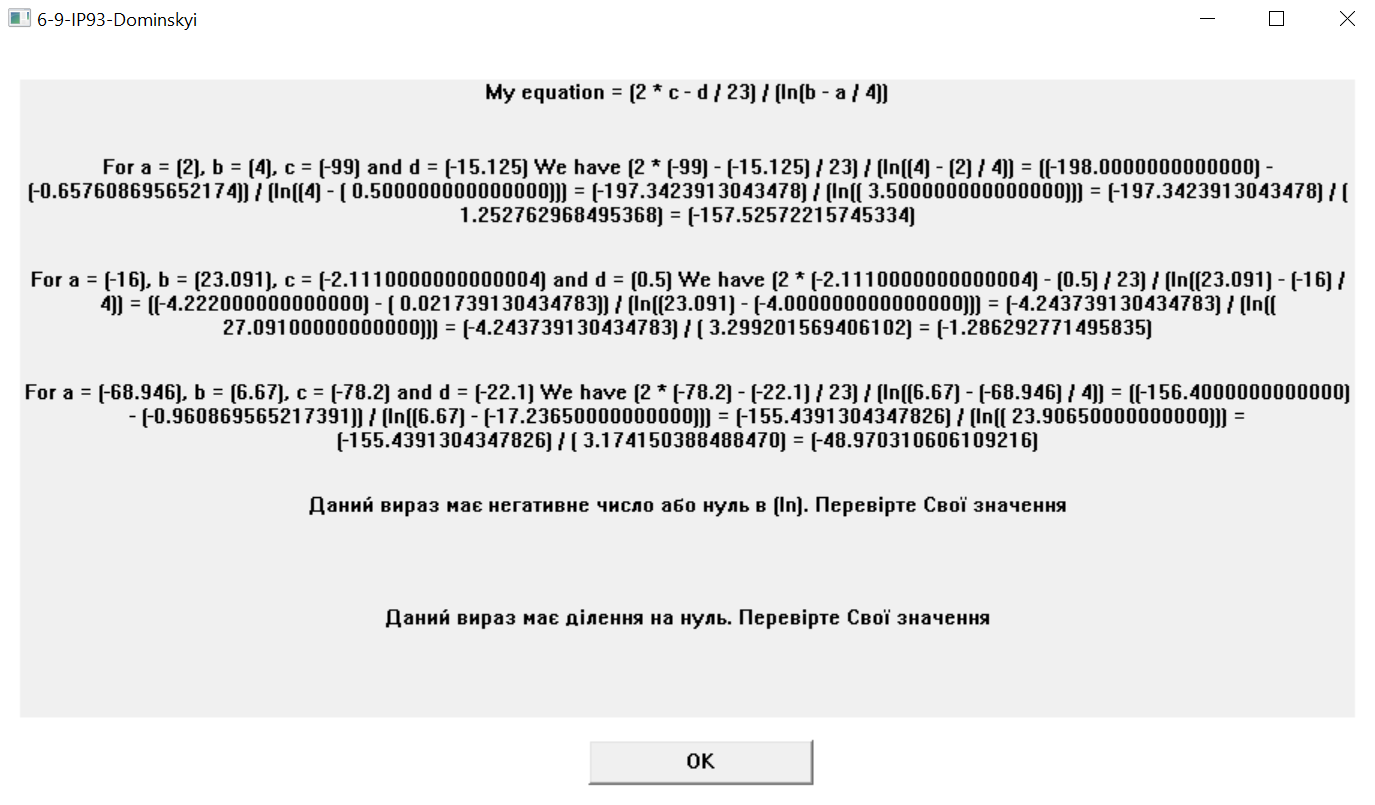
**Хід роботи**

1 - 4:

Вікно з описом:



Вікно з виразами



5.

Номер у списку = 9

Варіант індивідуального завдання:

Спочатку проведемо обчислення з невідомими:

А тепер проведемо для кожних значень a, b, c:

1. a = 2.0, b = 4.0, c = -99.0, d = -15.125
2. a = -16.0, b = 23.091, c = -2.111, d = 0.5
3. a = -68.946, b = 6.67, c = -78.2, d = -22.1
4. a = 0.001, b = -3.33, c = 123.4, d = -9.0

Як видно з даних розрахунків, до ln потрапляє негативне число, що є неможливим для обрахування

1. a = 4.0, b = 2.0, c = 44.47, d = 12.2222

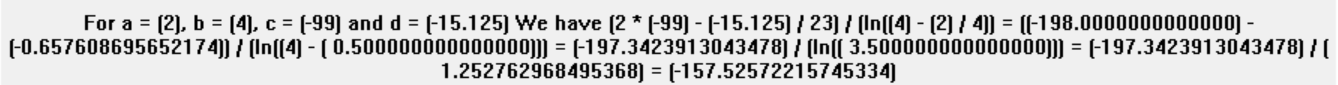
У даному прикладі до знаменнику потрапляє нуль, що є неможливим для обрахування

6.

А тепер давайте звіримо результати з вікна програми та результати, які Ми робили у пункті №5:

Вираз №1:

Результат у вікні:

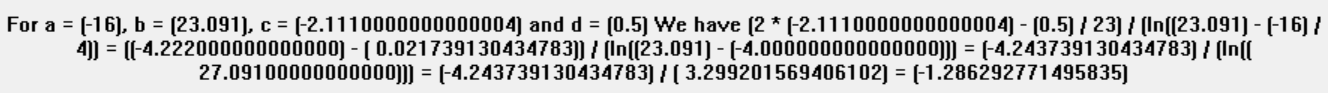


Та той, що Ми вирахували:

1. a = 2.0, b = 4.0, c = -99.0, d = -15.125

Вираз №2:

Результат у вікні:

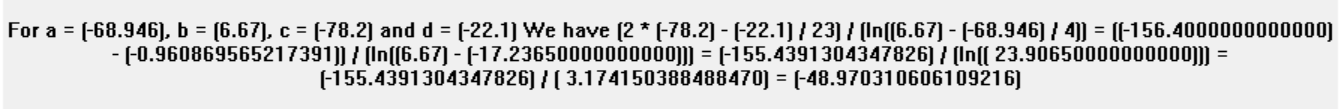


Та той, що Ми вирахували:

1. a = -16.0, b = 23.091, c = -2.111, d = 0.5

Вираз №3:

Результат у вікні:



Та той, що Ми вирахували:

1. a = -68.946, b = 6.67, c = -78.2, d = -22.1

Вираз №4:

Результат у вікні:



Та той, що Ми вирахували:

1. a = 0.001, b = -3.33, c = 123.4, d = -9.0

Як видно з даних розрахунків, до ln потрапляє негативне число, що є неможливим для обрахування

Вираз №5:

Результат у вікні:



Та той, що Ми вирахували:

5. a = 4.0, b = 2.0, c = 44.47, d = 12.2222

У даному прикладі до знаменнику потрапляє нуль, що є неможливим для обрахування.

Як бачимо, усі значення майже абсолютно точно зійшлися (різниця була помічена жовтим кольором. Це можна пояснити двома речами: тим,що обраховувати такі малі значення для комп’ютерів не так легко та тим, що деякі значення просто заокруглилися до певного знаку), отже Наша програма працює вірно

Лістинг програми:

; Processors

.386

.model **flat,** **stdcall**

option **CaseMap:None**

WinWarningProto proto **:dword,:dword,:dword**

WinMainProto proto **:dword,:dword,:dword**

; Libraries And Macroses

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

include \masm32\include\windows.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

; Our Macroses

; We place them here, 'cause it won't degrade the readability of the code

; Macros #1 for printing some text

PrintInformationInWindow macro heightPosition**,** infoToShow

; for example, this commentary is included into macroexpansion

;; but this - not

; just pass position of the text on vertical

; and text, that We want to show

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheText**,** offset infoToShow**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_TEXT **or** SS\_CENTER **or** BS\_VCENTER**,**

16**,** heightPosition**,** 690**,** 50**,**

hWnd**,** 7044**,** hInstance**,** NULL

endm

; Macros #2 for calculating

DoArithmeticOperations macro aInt**,** bInt**,** cInt

; Label for ending macros

Local EndThisMacros

; My equation = (21 - a \* c / 4) / (1 + c / a + b)

; check, if numerator aInt != 0

.if aInt **==** 0

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** ZeroDivisionText

**jmp** EndThisMacros

.else

; (1 + c / a + b)

; move cInt into al register

**mov** **al,** cInt

; Convert byte to word

**cbw**

; cInt / aInt

**idiv** aInt

; 1 + cInt / aInt

**add** **al,**1

; 1 + cInt / aInt + bInt

**add** **al,** bInt

; move (1 + cInt / aInt + bInt) into buffer TempPlaceForText

; check, if numerator al ((1 + c / a + b)) != 0

.if **al** **==** 0

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** ZeroDivisionText

**jmp** EndThisMacros

.endif

**mov** TempPlaceForText**,** **al**

; (21 - a \* c / 4)

; move aInt into al register

**mov** **al,** aInt

; Convert byte to word

**cbw**

; aInt \* cInt

**imul** cInt

; move 4 into bl

**mov** **bl,** 4

; aInt \* cInt / 4

**idiv** **bl**

; move 21 into bl

**mov** **bl,** 21

; 21 - (aInt \* cInt / 4)

**sub** **bl,** **al**

; Convert byte to word

**cbw**

; move bl into al

**mov** **al,bl**

; Convert byte to word

**cbw**

; (21 - a \* c / 4) / (1 + c / a + b)

**idiv** TempPlaceForText

; Convert byte to word

**cbw**

; Copies the contents of the source operand (register or memory location)

; to the destination operand (register) and sign extends the value to 16 or 32 bits

**movsx** **eax,** **al**

; move eax into intFinal

**mov** intAlmostFinal**,** **eax**

; Computes the bit-wise logical AND of first operand

; (source 1 operand) and the second operand (source 2 operand)

; and sets the SF, ZF, and PF status flags according to the result.

; The result is then discarded.

**test** **al,** **al**

; Status of parity flag

; if odd, then multiply

.if **parity?**

;; move 5 into bl

**mov** **bl,** 5

;; Convert byte to word

**cbw**

;; al \* 5

**imul** **bl**

;; Convert byte to word

**cbw**

; Copies the contents of the source operand (register or memory location)

; to the destination operand (register) and sign extends the value to 16 or 32 bits

**movsx** **eax,** **al**

; move eax into intFinal

**mov** intFinal**,** **eax**

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** equationVariablesForOdd**,**

intA**,** intB**,** intC**,** intA**,** intC**,** intC**,** intA**,** intB**,** intAlmostFinal**,** intFinal

; if even, then divide

.else

;; move 2 into bl

**mov** **bl,** 2

;; Convert byte to word

**cbw**

; al / 2

**idiv** **bl**

;; Convert byte to word

**cbw**

; Copies the contents of the source operand (register or memory location)

; to the destination operand (register) and sign extends the value to 16 or 32 bits

**movsx** **eax,** **al**

; move eax into intFinal

**mov** intFinal**,** **eax**

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** equationVariablesForEven**,**

intA**,** intB**,** intC**,** intA**,** intC**,** intC**,** intA**,** intB**,** intAlmostFinal**,** intFinal

.endif

.endif

EndThisMacros**:**

endm

.data?

hInstance HINSTANCE **?** ; Handle of our program

hWndOfWarnWindow HWND **?** ; Handle of our warn window

hWndOfMainWindow HWND **?** ; Handle of our main window

;; Text, that We will show

TempPlaceForText DB 256 DUP**(?)**

; Data Segment

.data

StartingText DB "У наступному вікні Ви побачите 5 різних арифметичних виразів"**,** 13**,** 0

ZeroDivisionText DB "Даний вираз має ділення на нуль. Перевірте Свої значення"**,** 13**,** 0

; Name Of Message Box

MsgBoxName DB "5-9-IP93-Dominskyi"**,** 0

NameOfTheWarnWindows DB "Window with warn text"**,** 0 ; the name of our warn window class

NameOfMainWindows DB "Window with main text"**,** 0 ; the name of our success window class

NameOfTheButton DB "Button"**,** 0 ; the name of our button class

NameOfTheText DB "Static"**,** 0 ; the name of our text class

TextForOKButton DB "ОК"**,** 0

; My equation = (21 - a \* c / 4) / (1 + c / a + b)

; can't be 1 or 0

; first way of declaring array

IntegersA DB 2**,** 8 **,** **-**6**,** **-**2**,** 10 ;; first numbers

IntegersB DB **-**33**,** 23**,** **-**2**,** 8**,** **-**3 ;; second numbers

; and the second one

IntegersC DB 66 ;; third numbers

DB 24

DB **-**12

DB **-**2

DB 10

;; global variables for interpolating for main window

;; (I will put some int into them and show in main window)

;; mostly used for negative nums

intA DD 0

intB DD 0

intC DD 0

intAlmostFinal DD 0

intFinal DD 0

; for automating

possibleHeight DD 12

coefficientOfMultiplyingForTextHeight DD 3

; first text to show

variantToShow DB "My equation = (21 - a \* c / 4) / (1 + c / a + b)"**,** 13**,** 0

; forms, which I will be filling with variables

equationVariablesForOdd DB "For a = (%d), b = (%d) and c = (%d) We have (21 - (%d) \* (%d) / 4) / (1 + (%d) / (%d) + (%d)) = (%d) \* 5 = (%d)"**,** 13**,** 0

equationVariablesForEven DB "For a = (%d), b = (%d) and c = (%d) We have (21 - (%d) \* (%d) / 4) / (1 + (%d) / (%d) + (%d)) = (%d) / 2 = (%d)"**,** 13**,** 0

; Code Segment

.code

start**:** ; Generates program start-up code

invoke WinWarningProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

invoke GetModuleHandle**,** NULL

**mov** hInstance**,** **eax**

invoke WinMainProto**,** hInstance**,**NULL**,** SW\_SHOWDEFAULT ;invoke function

invoke ExitProcess**,** **eax** ; quit program. code returns in EAX register from Main Function.

; function declaration of WinWarn

WinWarningProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndWarnProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**1

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfTheWarnWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfTheWarnWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

470**,** 310**,** 300**,** 150**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfWarnWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinWarningProto endp

; function declaration of WinSuccess

WinMainProto proc hInst**:**HINSTANCE**,**hPrevInst**:**HINSTANCE**,**CmdShow**:dword**

; there we need LOCAL variables

LOCAL wc**:**WNDCLASSEX

LOCAL msg**:**MSG

LOCAL hwnd**:**HWND

; assign variables of WNDCLASSEX

; window class is a specification of a window

**mov** wc.cbSize**,** sizeof WNDCLASSEX

**mov** wc.style**,** CS\_HREDRAW **or** CS\_VREDRAW

**mov** wc.lpfnWndProc**,** offset WndMainProc

**mov** wc.cbClsExtra**,** NULL

**mov** wc.cbWndExtra**,** NULL

**push** hInstance

**pop** wc.hInstance

**mov** wc.hbrBackground**,** COLOR\_WINDOW**+**1

**mov** wc.lpszMenuName**,** NULL

**mov** wc.lpszClassName**,** offset NameOfMainWindows

invoke LoadIcon**,** NULL**,** IDI\_APPLICATION

**mov** wc.hIcon**,** **eax**

**mov** wc.hIconSm**,** **eax**

invoke LoadCursor**,** NULL**,** IDC\_ARROW

**mov** wc.hCursor**,** **eax**

; create class of the window

invoke RegisterClassEx**,** **addr** wc

invoke CreateWindowEx**,** NULL**,**

offset NameOfMainWindows**,**

offset MsgBoxName**,**

WS\_OVERLAPPEDWINDOW **or** DS\_CENTER**,**

250**,** 200**,** 740**,** 300**,**

NULL**,** NULL**,** hInst**,** NULL

**mov** hWndOfMainWindow**,** **eax**

; write window handle in eax

**mov** hwnd**,eax**

; Show window

invoke ShowWindow**,** hwnd**,**CmdShow

; update screen

invoke UpdateWindow**,** hwnd

; waits for message

.while TRUE

;returns FALSE IF WM\_QUIT message is received and will kill the loop

invoke GetMessage**,** **addr** msg**,**NULL**,**0**,**0

.break .IF **(**!**eax)**

;takes raw keyboard input and generates a new message

invoke TranslateMessage**,** **addr** msg

;sends the message data to the window procedure responsible for the specific window the message is for

invoke DispatchMessage**,** **addr** msg

; end while

.endw

; code returns in EAX register from Main Function.

**mov** **eax,** msg.wParam

; return

**ret**

;The ENDP directive defines the end of the procedure

;and has the same name as in the PROC directive

WinMainProto endp

WndMainProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

**mov** **edi,** 0

; invoke macros #1 one time to create text

PrintInformationInWindow possibleHeight**,** offset variantToShow

;; do the loop

LoopItself**:**

;; mov int with sign extending from first array into eax

**movsx** **eax,** IntegersA**[edi]**

;; mov eax with sign extending into global variable

**mov** intA**,** **eax**

;; mov int with sign extending from second array into eax

**movsx** **eax,** IntegersB**[edi]**

;; mov eax with sign extending into global variable

**mov** intB**,** **eax**

;; mov int with sign extending from third array into eax

**movsx** **eax,** IntegersC**[edi]**

;; mov eax with sign extending into global variable

**mov** intC**,** **eax**

;; start macros with ints from arrays

DoArithmeticOperations IntegersA**[edi],** IntegersB**[edi],** IntegersC**[edi]**

; mov possibleHeight into eax

**mov** **eax,** possibleHeight

;; Convert byte to word

**cbw**

; mov possibleHeight into ebx

**mov** **ebx,** coefficientOfMultiplyingForTextHeight

;; Convert byte to word

**cbw**

;; coefficientOfMultiplyingForTextHeight \* possibleHeight

;; eax \* ebx

**imul** **ebx**

**imul** **esi**

; print text

PrintInformationInWindow **eax,** offset TempPlaceForText

**inc** **edi**

**inc** **esi**

**cmp** **edi,** 5

**jne** LoopItself

; create button

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForOKButton**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

295**,** 215**,** 150**,** 30**,**

hWnd**,** 7033**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,** **eax**

**ret**

WndMainProc endp

WndWarnProc proc hWnd**:**HWND**,** ourMSG**:**UINT**,** wParam**:**WPARAM**,** lParam**:**LPARAM

; on window close

.IF ourMSG**==**WM\_CLOSE

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSEIF ourMSG**==**WM\_CREATE

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheButton**,** offset TextForOKButton**,**

WS\_CHILD **or** WS\_VISIBLE **or** BS\_CENTER **or** BS\_TEXT **or** BS\_VCENTER**,**

65**,** 65**,** 150**,** 30**,**

hWnd**,** 7003**,** hInstance**,** NULL

invoke CreateWindowEx**,**NULL**,**

offset NameOfTheText**,** offset StartingText**,**

WS\_VISIBLE **or** WS\_CHILD **or** BS\_TEXT **or** SS\_CENTER **or** BS\_VCENTER**,**

16**,** 10**,** 250**,** 50**,**

hWnd**,** 7004**,** hInstance**,** NULL

.ELSEIF ourMSG**==**WM\_COMMAND

; exit program

invoke DestroyWindow**,**hWnd

invoke PostQuitMessage**,**NULL

.ELSE

; process the message

invoke DefWindowProc**,**hWnd**,**ourMSG**,**wParam**,**lParam

**ret**

.ENDIF

ExitCode**:**

**xor** **eax,eax**

**ret**

WndWarnProc endp

end start

**Висновок:**

Я навчився працювати з одновимірними масивами, командами для різних арифметичних операцій, таких як:ділення, множення, додавання та віднімання. Також дізнався про команди CBW, MOVSX, за допомогою яких набагато легше працювати з числами, які мають знаки