**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

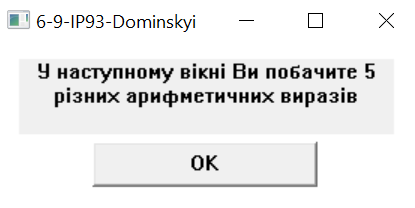
* Значення змінних a, b, c та d повинні мати бути дійсними числами, не рівними 0 або 1, допустимими для обчислення трансцендентних функцій, а тригонометричні функції обчислюються від аргументів, заданих в радіанах

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.

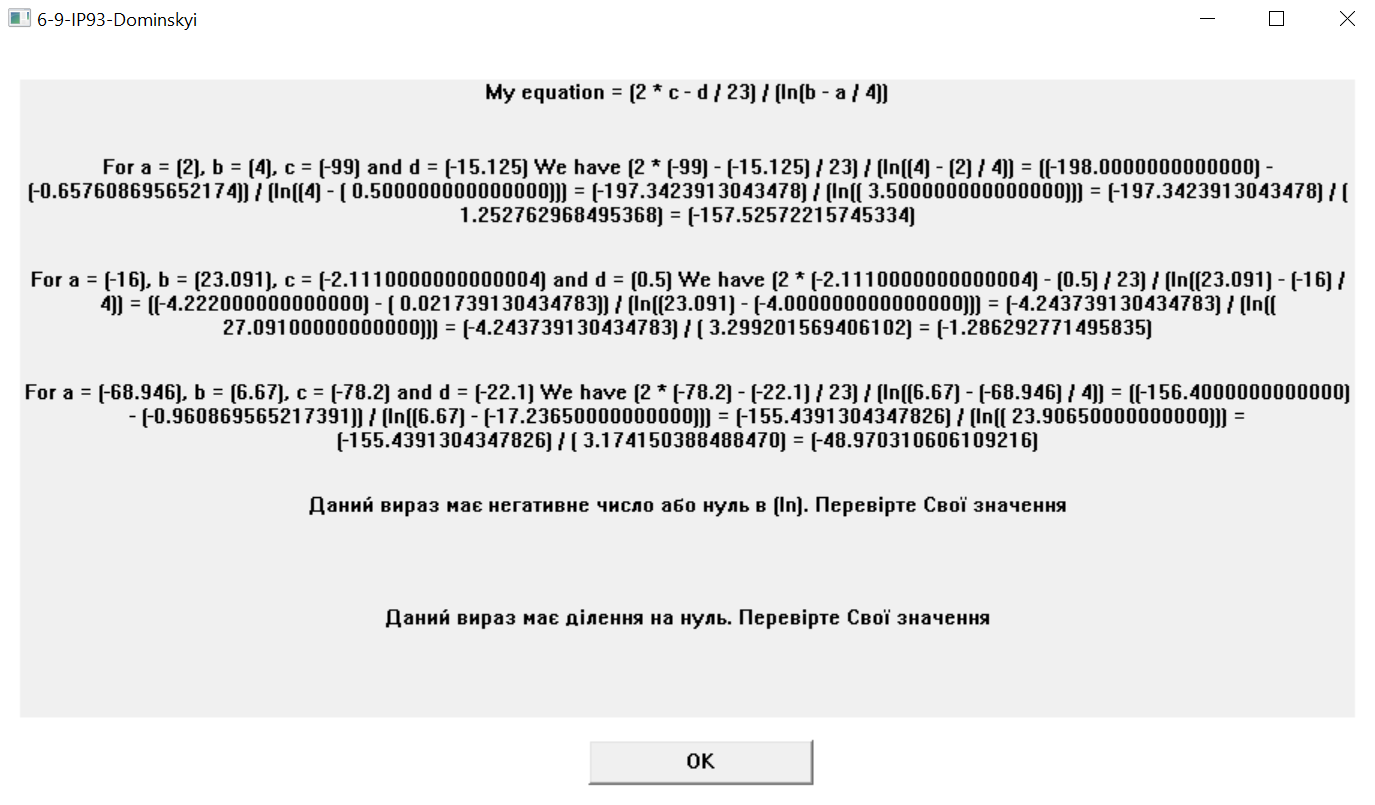
**Хід роботи**

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

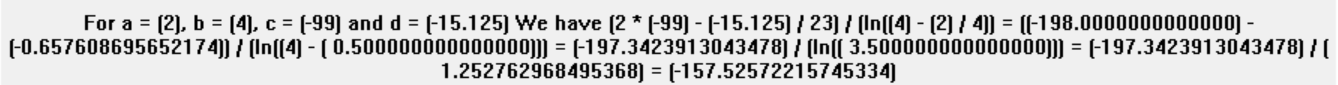
У даному прикладі до знаменнику потрапляє нуль, що є неможливим для обрахування

6.

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

Вираз №1:

Результат у вікні:

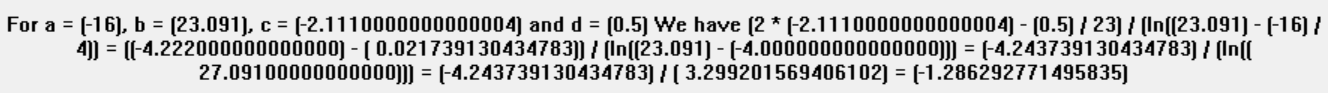


Та той, що Ми вирахували:

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

Вираз №2:

Результат у вікні:

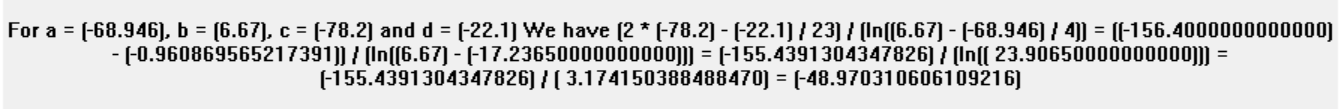


Та той, що Ми вирахували:

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**/**Fpu.lib

include **/**masm32**/**include**/**Fpu.inc

include **/**masm32**/**include**/**masm32rt.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**,** 890**,** 75**,**

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

endm

; Macros #2 for calculating

DoArithmeticOperations macro aFloat**,** bFloat**,** cFloat**,** dFloat

; Label for ending macros

Local EndThisMacros

; Label for checking, if numerator is zero (for zero division)

Local NumberIsZero

; Label for checking, if numerator is zero or less than it (for ln function)

Local NumberIsLessOrZero

; My equation = (2 \* c - d / 23) / (ln(b - a / 4))

**finit** ; FPU Initialization

; 2 \* c

; move 2 into st(0)

**fld** firstConstant

; move c into st(0) and 2 into st(1)

**fld** cFloat

; multiply 2 by c and move result into st(0)

**fmul**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferTwoMulC**,** SRC1\_FPU **or** SRC2\_DIMM

; d / 23

; move d into st(0) and 2\*c into st(1)

**fld** dFloat

; move 23 into st(0), d into st(1) and 2\*c into st(2)

**fld** secondConstant

; divide d by 23 and move result into st(0), 2\*c to st(1)

**fdiv**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferDdivTwenThree**,** SRC1\_FPU **or** SRC2\_DIMM

; 2 \* c - d / 23

; subtract d/23 from 2\*c, move result into st(0)

**fsub**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferFirstPart**,** SRC1\_FPU **or** SRC2\_DIMM

; move ln(2) into st(0) and 2\*c-d/23 into st(1)

**fldln2**

; move b into st(0), ln(2) into st(1) and 2\*c-d/23 into st(2)

**fld** bFloat

; move a into st(0), b into st(1), ln(2) into st(2) 2\*c-d/23 into st(3)

**fld** aFloat

; move 4 into st(0), a into st(1), b into st(2), ln(2) into st(3) and 2\*c-d/23 into st(4)

**fld** thirdConstant

; divide a by 4 and move it into st(0), b into st(1), ln(2) into st(2) and 2\*c-d/23 into st(3)

**fdiv**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferAdivFour**,** SRC1\_FPU **or** SRC2\_DIMM

; subtract a/4 from b, move result into st(0), ln(2) into st(1) and 2\*c-d/23 into st(2)

**fsub**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferBsubPartOfLn**,** SRC1\_FPU **or** SRC2\_DIMM

; compare, if number is zero or less for ln

; compares the contents of st (0) to the source

**fcom** zero

; saves the current value of the SR register to the receiver

**fstsw** **ax**

; loads flags

**sahf**

; jump, if equal to zero

**je** NumberIsLessOrZero

; jump, if less than zero

**jb** NumberIsLessOrZero

; find ln(b - a/4) and move it into st(0), 2\*c-d/23 into st(1)

**fyl2x**

; convert float to text with 18 digits after "," into buffer

invoke FpuFLtoA**,** 0**,** 18**,** **addr** BufferSecondPart**,** SRC1\_FPU **or** SRC2\_DIMM

; compare, if number is zero for dividing

; compares the contents of st (0) to the source

**fcom** zero

; saves the current value of the SR register to the receiver

**fstsw** **ax**

; loads flags

**sahf**

; jump, if equal to zero.zero

**je** NumberIsZero

; compares the contents of st (0) to the source

**fcom** negativeZero

; saves the current value of the SR register to the receiver

**fstsw** **ax**

; loads flags

**sahf**

; jump, if equal to negative zero.zero

**je** NumberIsZero

; compares the contents of st (0) to zero

**ftst**

; saves the current value of the SR register to the receiver

**fstsw** **ax**

; loads flags

**sahf**

; jump, if equal to zero

**je** NumberIsZero

; divides 2\*c-d/23 by ln(b-a/4) and move it into st(0)

**fdiv**

; (2 \* c - d / 23) / (ln( b - a / 4))

; saves st(0) into variable

**fstp** floatFinal

;; value for final result

invoke FloatToStr2**,** floatFinal**,** **addr** BufferFloatFinal

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** equationVariables**,**

**addr** BufferFloatA**,** **addr** BufferFloatB**,** **addr** BufferFloatC**,** **addr** BufferFloatD**,**

**addr** BufferFloatC**,** **addr** BufferFloatD**,** **addr** BufferFloatB**,** **addr** BufferFloatA**,**

**addr** BufferTwoMulC**,** **addr** BufferDdivTwenThree**,** **addr** BufferFloatB**,**

**addr** BufferAdivFour**,** **addr** BufferFirstPart**,** **addr** BufferBsubPartOfLn**,**

**addr** BufferFirstPart**,** **addr** BufferSecondPart**,** **addr** BufferFloatFinal

**jmp** EndThisMacros

NumberIsZero**:**

;; parsing variables into TempPlaceForText

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

**jmp** EndThisMacros

NumberIsLessOrZero**:**

;; parsing variables into TempPlaceForText

invoke wsprintf**,** **addr** TempPlaceForText**,** **addr** NegativeOrZeroLnText

**jmp** EndThisMacros

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**(?)**

; Buffers for final float numbers

BufferFloatA DB 32 DUP**(?)**

BufferFloatB DB 32 DUP**(?)**

BufferFloatC DB 32 DUP**(?)**

BufferFloatD DB 32 DUP**(?)**

BufferFloatFinal DB 32 DUP**(?)**

; Start = (2 \* c - d / 23) / (ln(b - a / 4))

; Buffers for intermediate results

; First Step

; Value of 2 \* c

BufferTwoMulC DB 32 DUP**(?)**

; Value of d / 23

BufferDdivTwenThree DB 32 DUP**(?)**

; Value of a - 4

BufferAdivFour DB 32 DUP**(?)**

; Second Step

; Value of 2 \* c - d / 23

BufferFirstPart DB 32 DUP**(?)**

; Value of b - a / 4

BufferBsubPartOfLn DB 32 DUP**(?)**

; Third Step

; Value of ln(b - a / 4)

BufferSecondPart DB 32 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

NegativeOrZeroLnText DB "Даний вираз має негативне число або нуль в (ln). Перевірте Свої значення"**,** 13**,** 0

; Name Of Message Box

MsgBoxName DB "6-9-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 = (2 \* c - d / 23) / (ln( b - a / 4))

; can't be 1 or 0

; first way of declaring array

FloatsA dq 2.0**,** **-**16.0**,** **-**68.946**,** 0.001**,** 4.0 ;; first numbers

FloatsB dq 4.0**,** 23.091**,** 6.67**,** **-**3.33**,** 2.0 ;; second numbers

FloatsC dq **-**99.0**,** **-**2.111**,** **-**78.2**,** 123.4**,** 44.47 ;; third numbers

; and the second one

FloatsD dq **-**15.125 ;; fourth numbers

dq 0.5

dq **-**22.1

dq **-**9.0

dq 12.2222

firstConstant dq 2.0

secondConstant dq 23.0

thirdConstant dq 4.0

zero dq 0.0

negativeZero dq **-**0.0

;; global variables for interpolating for main window

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

;; mostly used for negative nums

floatFinal DQ 0

; for automating

possibleHeight DD 25

coefficientOfMultiplyingForTextHeight DD 3

valueForDQvalues DB 8

; first text to show

variantToShow DB "My equation = (2 \* c - d / 23) / (ln(b - a / 4))"**,** 13**,** 0

; form, which I will be filling with variables

equationVariables DB "For a = (%s), b = (%s), c = (%s) and d = (%s) We have (2 \* (%s) - (%s) / 23) / (ln((%s) - (%s) / 4)) = ((%s) - (%s)) / (ln((%s) - (%s))) = (%s) / (ln((%s))) = (%s) / (%s) = (%s)"**,** 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**,**

170**,** 50**,** 940**,** 550**,**

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**:**

;; values for equation

invoke FloatToStr2**,** FloatsA**[**8**\*edi],** **addr** BufferFloatA

invoke FloatToStr2**,** FloatsB**[**8**\*edi],** **addr** BufferFloatB

invoke FloatToStr2**,** FloatsC**[**8**\*edi],** **addr** BufferFloatC

invoke FloatToStr2**,** FloatsD**[**8**\*edi],** **addr** BufferFloatD

;; start macros with floats from arrays

DoArithmeticOperations FloatsA**[**8**\*edi],** FloatsB**[**8**\*edi],** FloatsC**[**8**\*edi],** FloatsD**[**8**\*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**,**

395**,** 465**,** 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 з плаваючої комою, таких як:ділення, множення, додавання та віднімання у математичному співпроцесорі. Попрацював з різними форматами даних