# Лабораторна робота **№1**

**Засоби розробки програм на Assembler**

***Мета роботи:***набуття навичок створення програм на Microsoft Macro Assembler (MASM).

***Завдання:*** Виконати та описати етапи створення у середовищі Microsoft Visual Studio програми на Microsoft Assembler (MASM), що відкриває вікно “Assembler Program“ з повідомленням “Hello, XX XX (XX XX – ім’я та призвіще студента)!”.

***Розв’язок:***

1) Запуск середовища Visual Studio (..\Microsoft Visual Studio\2017\Enterprise\Common7\IDE\devenv.exe). Після запуску на екрані з’явиться вікно зі стартовою сторінкою (рис. 1.1).

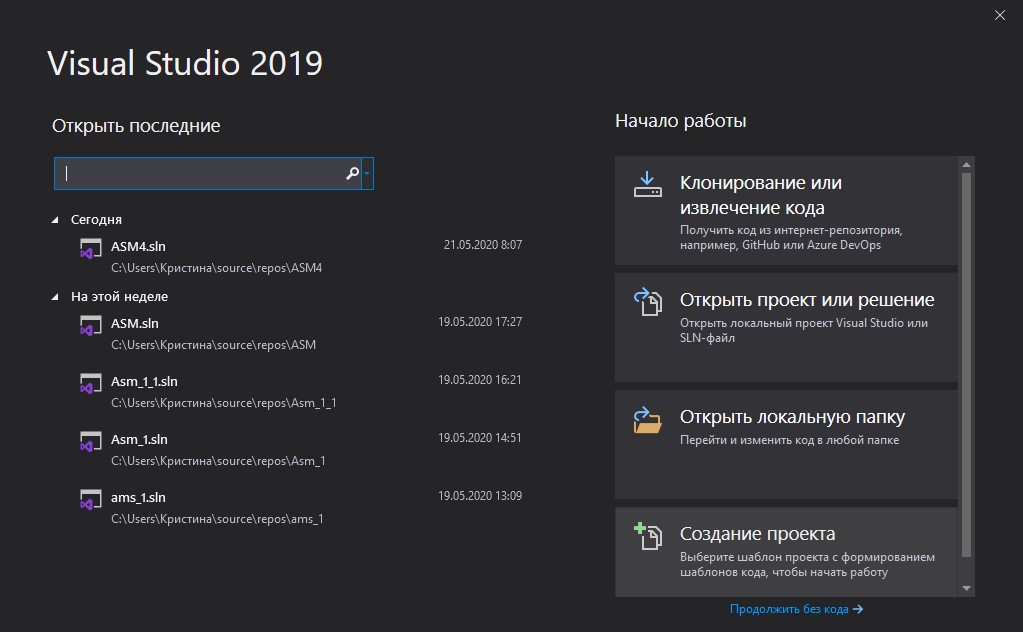


Рис. 1.1 – Стартова сторінка у середовищі Microsoft VS

2) Для створення проекту необхідно натиснути (Файл –> Создать –> Проект) (рис. 1.2).

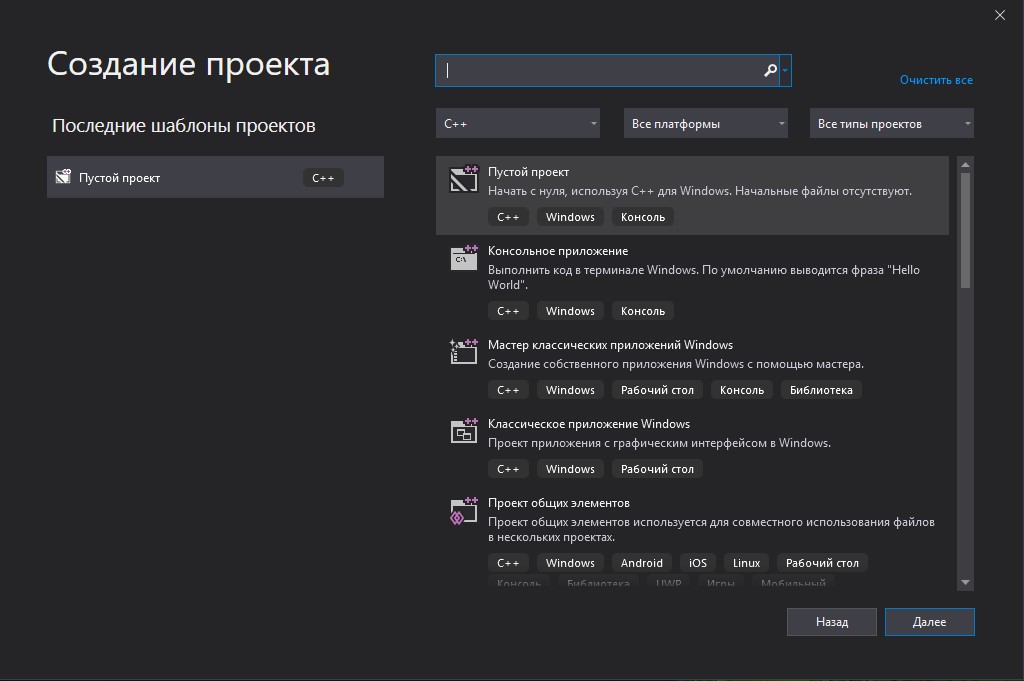


Рис. 1.2 – Створення проекту у середовищі Microsoft VS

3) Відкриється вікно «Создать проект», у якому вибираємо «Пустой проект» та вводимо ім’я проекту (Наприклад, *asm\_lab1\_1*). Ці дії виконані на рис. 1.3.

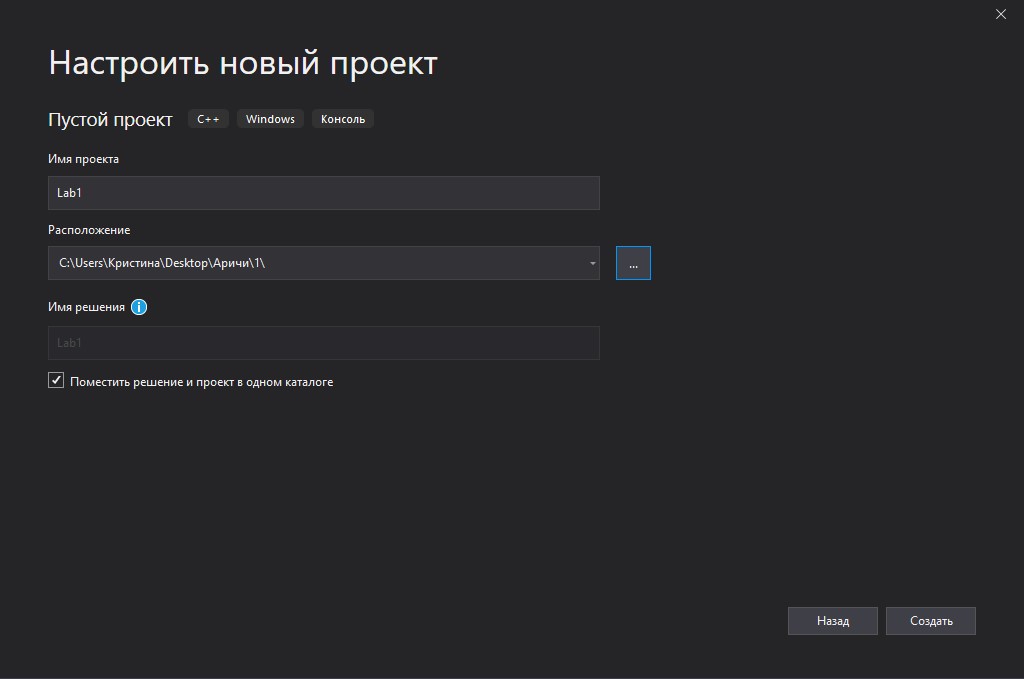


Рис. 1.3 – Створення проекту у середовищі Microsoft VS (продовження)

4) У вкладці «Обозреватель решений» по каталогу «Исходные файлы» натистаємо правою кнопою миші та із контексного меню вибираємо (Добавить –> Создать элемент...) – див. рис. 1.4.

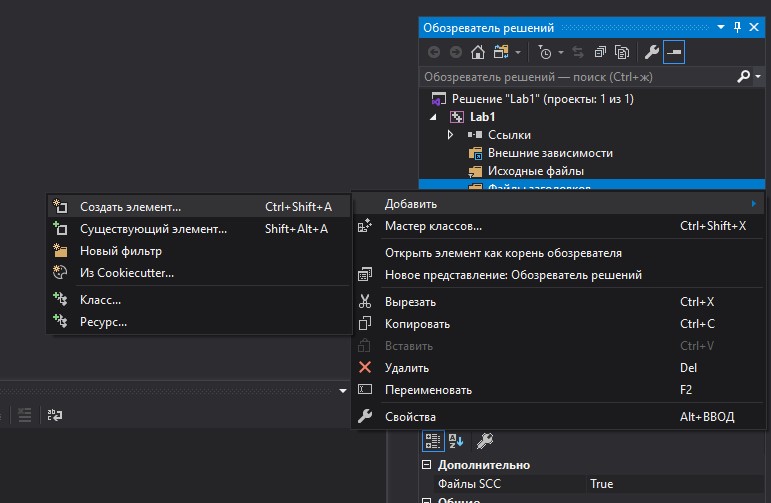


Рис. 1.4 – Додавання нового елементу в проект у середовищі Microsoft VS

5) У вікні «Добавление нового элемента» вводимо ім’я файлу та змінюємо розширення з .cpp на .asm (наприклад, lr01\_v01.asm) та натискаємо кнопку «Добавить» – див. рис. 1.5.

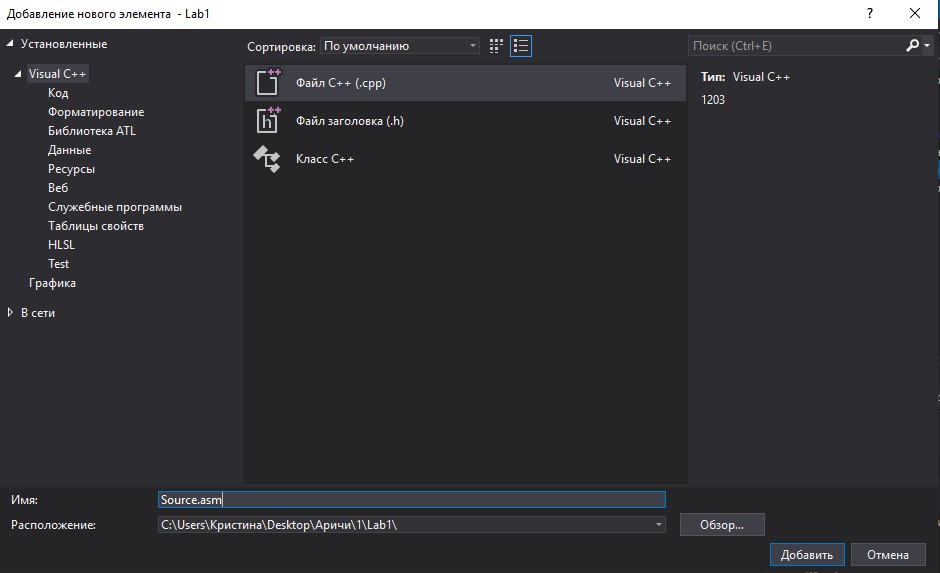


Рис. 1.5 – Додавання нового елементу в проект у середовищі Microsoft VS

6) Відкривається вікно текстового редактору для редагування цього файлу, де й вводимо наступний код (див. рис. 1.6):

.686P

.MODEL FLAT, STDCALL

.STACK 4096

.DATA

MB\_OK EQU 0

STR1 DB "Assembler program", 0

STR2 DB "Hello, Artur Boyn!", 0

HW DD ?

EXTERN MessageBoxA@16:NEAR

.CODE

START:

PUSH MB\_OK

PUSH OFFSET STR1

PUSH OFFSET STR2

PUSH HW

CALL MessageBoxA@16

RET

END START

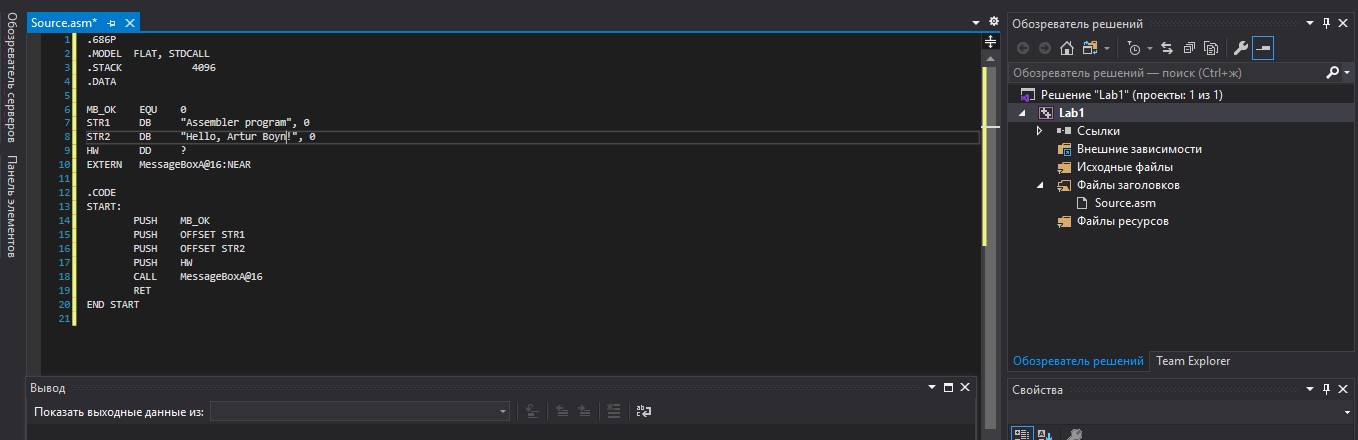


Рис. 1.6 – Введення тексту Source.asm-програми у середовищі Microsoft VS\

7) Натиснути ЛКМ на проект (на рис. 1.6 це – **ASM**) та натиснути ПКМ (Зависимости сборки –> Настройки сборки…). Відкриється вікно «Файлы настройки сборки для Visual C++», де ставиму галочку навпроти masm(.targets, .props) та напискаємо «ОК» (рис. 1.7).

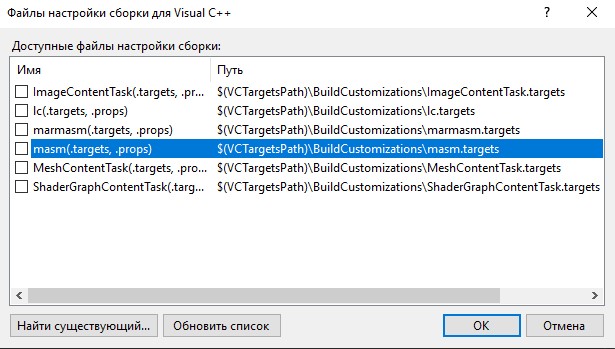


Рис. 1.7 – Налагодження зборки проекту в середовищі Microsoft VS

8) Наступний крок – ПКМ по файлу \*.asm (на рис. 1.6 це – **Source.asm**) –> Свойства. У полі «Тип элемента» виставляємо Microsoft Macro Assembler та натискаємо «ОК» – див. рис. 1.8.

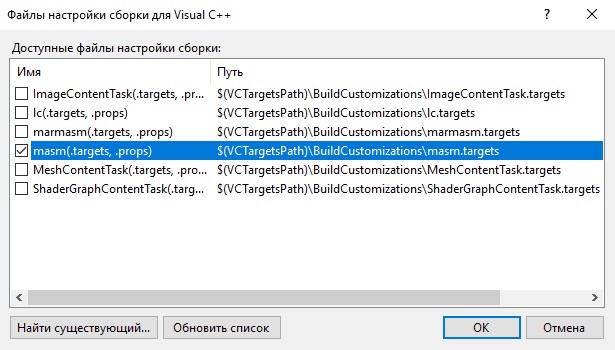


Рис. 1.8 – Налагодження зборки проекту в середовищі Microsoft VS

9) Натискаємо “Проект (на рис. 1.6 це – **ASM**)” –> Свойтва –> Свойтва конфигурации –> Компоновщик –> Система). У полі «Подсистема» ставимо Консоль (/SUBSYSTEM:CONSOLE) та натискаємо «ОК» (див. рис. 1.9).

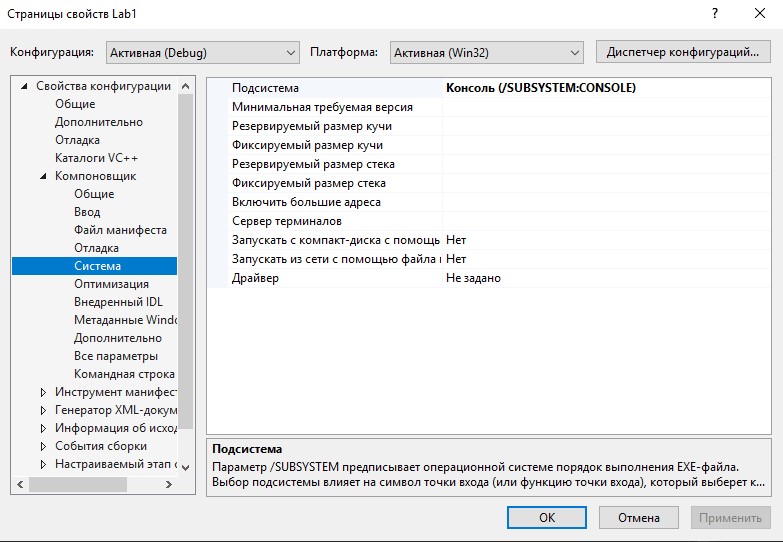


Рис. 1.9 – Налагодження зборки проекту в середовищі   
Microsoft VS (продовження)

10) Натискаємо «F5» для зборки та запуску програми (результати див. на рис. 1.10).

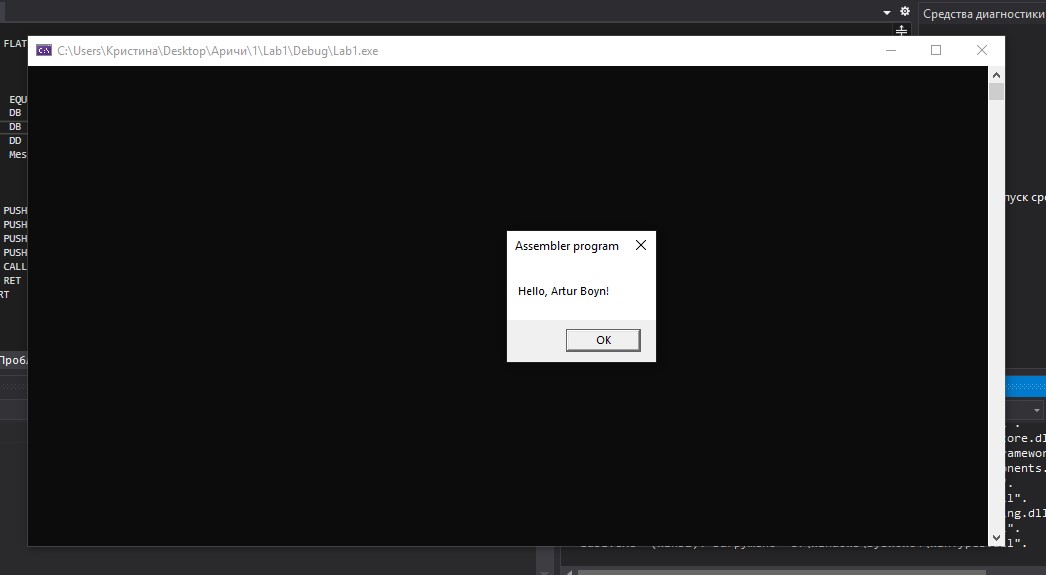


Рис. 1.10 – Результати роботи програми

#### Лабораторна робота №2.

**Розробка простих програм на Assembler**

***Мета роботи*:** Вивчення основних арифметичних команд та команд передачі даних; набуття навичок застосування цих команд для програмування на MASM 32- розрядних додатків.

***Теоретичні основи:*** Microsoft Macro Assembler (MASM) — асемблер для архітектури x86 під операційну систему Microsoft Windows, що використовує синтаксис Intel. Поточна версія MASM існує у двох різновидах: для 16- і 32-бітного коду і для 64-бітного (ML64).

До версії 6.12 був окремим продуктом Microsoft. Зараз він включений до складу різних компіляторів і середовищ розробки компанії.

Перші версії MASM з’явилися у 1981 році. До версії 5.0 MASM був доступний тільки для операційних систем MS DOS. Версії 5.1 і 6.0 розроблялися крім MS DOS також і для OS/2.

У 1999 році представила набір інструкцій SIMD і MMX, які невдовзі після цього став підтримувати і MASM. З випуском версії 6.15, Microsoft припинила підтримку MASM як окремого продукту і включила його до інструментарію Visual Studio. Підтримки 64-бітного режиму в MASM не було до випуску Visual Studio 2005, де MASM отримав номер версії 8.0. Сьогодні MASM продовжує використовуватися на платформі Win32, незважаючи на конкуренцію з такими новими продуктами, як NASM, FASM, TASM, HLASM.

**Особливості MASM**

У пакеті MASM спочатку робилися спроби поєднати зручність засобів програмування, властивих мовам високого рівня, з традиційними засобами машинно-орієнтованих мов. Наприклад, буква «М» в назві пакету означає слово «macro», тобто можливість створення макроозначень (або макросів), що представляють собою іменовані групи команд. Їх можна вставляти в програму в будь-якому місці, вказавши тільки ім'я групи. Колись така можливість дійсно була відмітною властивістю цього пакета, але зараз цим нікого не здивуєш - макрозасоби є у всіх асемблерах для платформи Intel (наприклад, TASM), але, тим не менш, назва залишилася.

За допомогою пакета MASM розробка програм виконується традиційним для асемблерного програмування способом - запуском окремих програм трансляції, компонування і налагодження. Для цього використовуються програми *masm.exe*, *mL.exe*, *link.exe* і *cv.exe*. Щоб уникнути плутанини необхідно відзначити, що трансляція вихідного файлу може здійснюватися двома програмами: masm.exe і *mL.exe*. В чому різниця? До виходу MASM версії 5.1 включно програма masm.exe була самостійним транслятором асемблера. Починаючи з MASM версії 6.0 трансляція асемблерного файлу зазвичай виконується програмою mL.exe, яка крім трансляції файлу викликає компонувальник *link.exe*. Ця зміна зроблено з метою уніфікації викликів компіляторів командного рядка для всіх підтримуваних цією фірмою мов програмування. Наприклад, компіляцію програми на мові С з командного рядка виконує програма *cl.exe*. Її завдання - компіляція вихідної програми на мовах C / C ++ і, при відсутності синтаксичних помилок, компонування і формування виконуваного модуля. Аналогічні завдання вирішує і програма *mL.exe.* Хоча в пакеті MASM 6.13 допустимо використання програми masm.exe, потрібно мати на увазі, що запуск *link.exe* в цьому випадку також повинен виконуватися окремо. Наявність програми masm.exe в пакеті MASM 6.13 можна пояснити міркуваннями сумісності з попередніми версіями пакета, тому особливого сенсу в її використанні немає.

**Арифметичні команди процесора** обмежені додавання, відніманням, множенням і діленням беззнакових та знакових цілих чисел. Дані команди зображено в табл. 6.1 – 6.5.

**Базові арифметичні команди**

|  |  |  |
| --- | --- | --- |
| Мнемоніка | Операнд(и) | Опис |
|  |  | Додавання байтів або слів |
|  |  | Віднімання байтів або слів |
|  |  | Беззнакове множення |
|  |  | Беззнакове множення |

Команда перемножує два беззнакових значення, при чому операнд команди є регістром загального призначення, або коміркою пам’яті. Розмір множника визначає розмір множеного і добутку, так якщо множник має розмір в байт, тоді множене в регістрі , а добуток записується в регістр Якщо множене , тоді множене знаходиться в регістрі , а результат в

Команда застосовується для ділення двох беззнакових значень, операнд команди є дільником, ділений регістр визначається розміром дільника. Коли дільник розміром в , ділене знаходиться в регістрі , частка розміщується в , а остача в . Коли дільник розміром в , ділене в регістрі , частка в а остача в

**Команди розмноження даних**

|  |  |
| --- | --- |
| Мнемоніка | Опис |
|  | Розширення |
|  | Розширення |

***Завдання:*** Розробити на MASM в IDE Microsoft Visual Studio програму, яка обчислює за формулою та виводить на екран значення X.



де *a*=100; *b*=3; *c*=2; *d*=4

***Розв’язок:***

Програма

.686P

.MODEL FLAT, STDCALL

.STACK 4096

.DATA

MB\_OK EQU 0

m EQU 95

str\_format DB "X = %d",0

str\_buffer DB 256 dup (0)

m\_title DB "Lab #2",0

A DB 100 ;a = 100

B DB 3 ;b = 3

C1 DB 2 ;c = 2

D DB 4 ;d = 4

X DD ?

extern MessageBoxA@16:NEAR

extern wsprintfA:NEAR

.CODE

Start:

xor eax, eax ; ax = 0

mov al, B ; al = b

mul al ; al = b ^ 2

mov bl, al ; bl = al = b ^ 2

xor eax, eax ; ax = 0

mov al, C1 ; al = c

mul al ; al = c ^ 2

mul bl ; ax = b ^ 2 \* c ^ 2

sub ax, 23d ; ax = (b ^ 2 \* c ^ 2) - 23

mov cx, ax ; cx = ax = (b ^ 2 \* c ^ 2) - 23

xor eax, eax ; ax = 0

mov al, D ; al = d

mul al ; al = d ^ 2

mov ah, B ; ah = b

sub al, ah ; al = d ^ 2 - b

mul A ; ax = (d ^ 2 - b) \* a

cwd

div cx ; ax = (d ^ 2 - b) \* a / (b ^ 2 \* c ^ 2) - 23

mov cx, ax ; cx = ax = (d ^ 2 - b) \* a / (b ^ 2 \* c ^ 2) - 23

xor eax, eax ; ax = 0

mov bl, B ; bl = b

cbw

add bx, cx ; bx = b + (ax = (d ^ 2 - b) \* a / (b ^ 2 \* c ^ 2) - 23)

cbw

sub bx, m ; bx = (bx = b + (ax = (d ^ 2 - b) \* a / (b ^ 2 \* c ^ 2) - 23)) - 95

MOV BYTE PTR[X], bl

PUSH X

PUSH OFFSET str\_format

PUSH OFFSET str\_buffer

CALL wsprintfA

ADD esp, 12

PUSH MB\_OK

PUSH OFFSET m\_title

PUSH OFFSET str\_buffer

PUSH 0

CALL MessageBoxA@16

RET

END Start

***Результати роботи****:* (див. рис. 2.1)

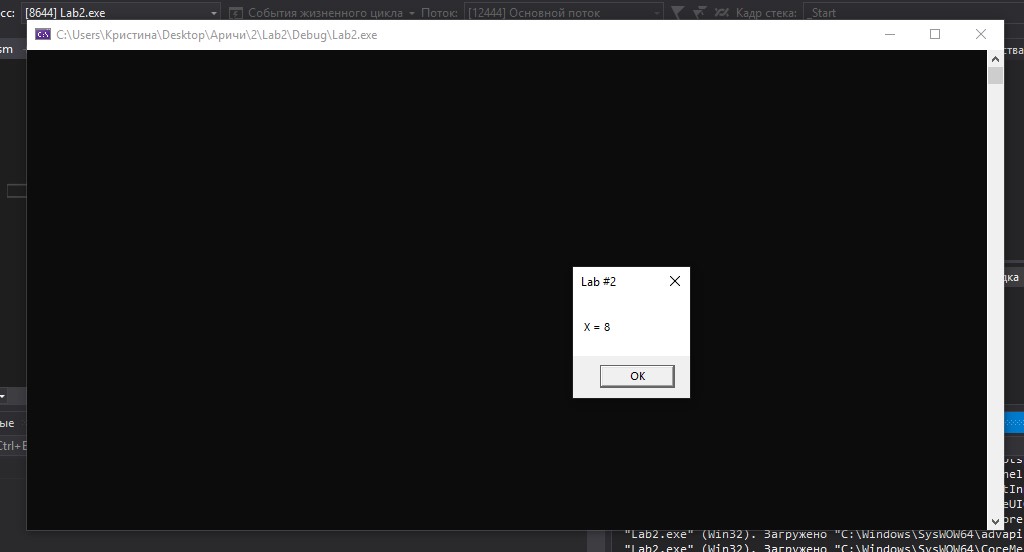


Рис. 2.1 – Результати роботи програми

**Висновок.**

Познайомились з методами створення програм на MASM, а також особливостями мови. Набули навичок створення програм на MASM з використанням інтегрованого середовища MSVS.

#### Лабораторна робота №3.

**Розробка структурованих програм на Assembler**

***Мета роботи*:** вивчення основних команд організації та використання процедур в програмах на Assembler; набуття навичок застосування цих команд під час розробки структурованих програм мовою MASM.

***Теоретичні основи:***

**Структурне програмування**

Структурне програмування — методологія програмування (модель конструювання програмного забезпечення).

Згідно з цією методологією будь-яка програма може бути створена використовуючи три конструкції:

* **послідовне виконання** — одноразове виконання операції в порядку запису їх (операцій) в тексті програми;
* **розгалуження** — виконання певної операції або декількох операцій в залежності від стану певної, наперед заданої умови;
* **цикл** — багаторазове виконання операції або групи операцій за умови виконання деякої наперед заданої умови. Така умова називається — умова продовження циклу.

Кожна конструкція являє собою блок із одним входом і одним виходом.

Структурована програма складається із вищеназваних блоків за двома правилами: **пакетування** (вихід одного блоку з'єднується із входом наступного) і **вкладення** (будь-який блок може бути замінений на структуру керування вибору або повторення).

Таким чином, структуровані програми містять всього сім типів структур керування, які з'єднуються всього двома способами.

Переваги:

* такі програми легко створювати та тестувати;
* скорочується термін розробки програми;
* полегшується супровід програми
* скорочення варіантів побудови програми, що знижує складність програмного забезпечення;
* логічно зв'язані операції знаходяться ближче один до одного, що полегшує аналіз алгоритму дозволяючи обходитись без блок-схем алгоритму (хоча наявність такої блок-схеми полегшує розуміння роботи алгоритму).

Недоліки:

* непропорційне зростання складності програми при збільшенні об'єму коду програми;
* складність створення паралельних програм.

**Структурне програмування на асемблері**

Одним з способів структурування програм на асемблері – це поділ програми на процедури. Процедура – окремий виконуючий блок, що виконує набір команд після виклику її з основної програми.

Процедура може бути незалежною від основної програми, або оперувати параметрами , що передаються в процедуру під час її виклику.

Оголошення процедури здійснюється після основного тіла програми за допомого службових слів – *PROC i END.*

Передача аргументів процедури може здійснюватись в декілька способів: через регістри, через стек.

При передачі параметрів через регістри, в основній програмі всі змінні, які будуть в подальшому використовуватись в процедурі копіюють в вільні регістри загального призначення: *ax, bx, cx, dx.* Процедура має доступ до цих регістрів незалежно від її розташування чи місця виклику. Вона може вільно ними оперувати.

Щоб передати параметри через стек, спочатку потрібно проштовхнути всі потрібні змінні в стек, якщо це можливо, або якщо ні – використовуючи в поміч регістри загального призначення.

Після виклику процедури на вершині стеку буде зберігатися адрес повернення з процедури. Спочатку потрібно зберегти цей адрес, а потім діставати всі потрібні параметри зі стеку, а при закінченні коду процедури, потрібно повернути на вершину стеку адрес повернення.

***Завдання***

Розробити на MASM в IDE Microsoft Visual Studio програму, яка обчислює за формулою та виводить на екран значення X.



де *a 100; b = 3; c = 2; d = 4;*

***Розв’язок:***

Програма

.686P

.MODEL FLAT, STDCALL

.STACK 4096

.DATA

a DD 100 ;a = 100

b DD 3 ;b = 3

c\_ DD 2 ;c = 2

d DD 4 ;d = 4

x DD ?

message DB "Lab#3", 0

STR1 DB 20 DUP(? )

HW DD ?

MB\_OK EQU 0

EXTERN MessageBoxA@16:NEAR

.CODE

Start:

xor eax, eax

xor ebx, ebx

xor ecx, ecx

push[a]

push[b]

push[c\_]

push[d]

call Expression

mov[x], eax

call PrintNum

push MB\_OK

push OFFSET message

push OFFSET STR1

push HW

call MessageBoxA@16

RET

Expression PROC

push ebp ; збереження вказівника

mov ebp, esp ; передача вказівника на вершину стека

xor eax, eax ; ax = 0

mov ax, [ebp+16] ; ax = b

mul ax ; ax = b ^ 2

mov cx, ax ; cx = ax = b ^ 2

xor eax, eax ; ax = 0

mov ax, [ebp+12] ; ax = c

mul ax ; ax = c ^ 2

mul cx ; ax = c ^ 2 \* b ^ 2

sub ax, 23d ; ax = (c ^ 2 \* b ^ 2) - 23

mov cx, ax ; cx = ax = (c ^ 2 \* b ^ 2) - 23

xor eax, eax ; ax = 0

mov ax, [ebp+8] ; ax = d

mul ax ; ax = d ^ 2

sub ax, [ebp+16] ; ax = d ^ 2 - b

mov bx, [ebp+20] ; bx = a

mul bx ; dx:ax = (d ^ 2 - b) \* a

div cx ; ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23

mov cx, ax ; cx = ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23 ---> cx

xor eax, eax ; ax = 0

mov bx, [ebp+16] ; bx = b

add bx, cx ; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23)

sub bx, 95d ; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23) - 95

mov ax, bx

pop ebp ; повернути вказівник на наступний після функції рядок

RET

Expression Endp

PrintNum PROC

local dk : byte

mov [dk], 10

xor ecx, ecx

M1:

div [dk]

add ah, 30h

push eax

cbw

inc ecx

test al, al

jnz M1

xor esi, esi

xor ebx, ebx

lea ebx, STR1

M2:

pop eax

mov[ebx], ah

inc ebx

loop M2

RET

PrintNum Endp

END Start

***Результати роботи****:* (див. рис. 3.1)

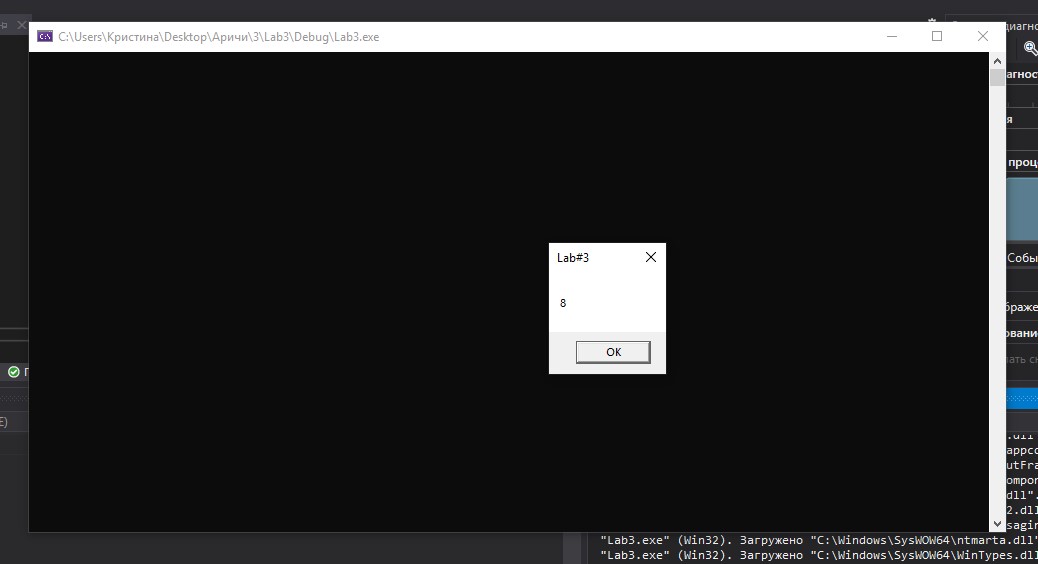


Рис. 3.1 – Результати роботи програми

**Висновок**

Познайомились з структурним програмуванням. Набули навичок розробки структурованих програм на MASM. Навчились використовувати різні методи передачі параметрів до процедури.

#### Лабораторна робота №4.

**Використання Assembler у мовах високого рівня**

***Мета роботи*:** вивчення основних можливостей використання асемблерного коду з мовами високого рівня; набуття навичок застосування зовнішніх процедур та вставок на асемблері при програмувані мовою С/С++.

**Теоретичні основи.**

Мова асемблера може служити для цілого ряду цілей, наприклад для підвищення швидкості роботи програми, скорочення споживання пам'яті та управління обладнанням. Вбудований код на асемблері можна використовувати для впровадження інструкцій мови асемблера безпосередньо в вихідні програми С і С ++ без додаткових кроків при збірці і компонуванні. Вбудований код на асемблері вбудований в компілятор, тому вам не потрібно окремий компонувальник для мови асемблера, такий як Microsoft Macro Assembler (MASM).

Програми з вбудованим кодом на мові асемблера не завжди можна перенести на інші апаратні платформи. Вбудований код на асемблері не підтримуються в процесорах ARM і x64.

**Асемблерна вставка**

У програмуванні, ассемблерной вставкою називають можливість компілятора вбудовувати низькорівневий код, написаний на асемблері, в програму, написану на мові високого рівня, наприклад, Сі або Ada.

Ключове слово *\_\_asm* ​​викликає вбудований асемблер і може відображатися всюди, де допустимо оператор С або С ++. Він не може відображатися самостійно. За ним повинна слідувати інструкція по збірці, група інструкцій, укладена в круглі дужки, або, в крайньому випадку, порожня пара круглих дужок. Термін **"*блок \_\_*asm"** в цьому розділі відноситься до будь-якої інструкції або групі інструкцій, в дужках або без них.

Підтримка стандартного ключового слова C ++ **asm** в Visual C ++ обмежена тим фактом, що компілятор не створить помилку за ключовим словом. Однак блок *asm* не створить ніякого значущого коду. Використовуйте *\_\_asm* ​​замість *asm*.

При використанні без круглих дужок ключове слово *\_\_asm* означає, що інша частина рядка - це оператор на мові збірки. При використанні з фігурними дужками воно означає, що кожен рядок між дужками - це оператор на мові збірки. Для забезпечення сумісності з попередніми версіями *\_asm* є синонімом *\_\_asm*.

Даний спосіб є елементом структурованого програмування, але в такому разі код написаний на різних мовах буде розміщений в одному виконавчому файлі. Цього можна уникнути використовуючи зовнішні функції, які оголошуються ключовим словом *extern.*

**Зовнішня функція**

Щоб використовувати зовнішню функцію на MASM, її потрібно описати в відокремленому файлі з розширенням *.asm* звичайним способом, але з єдиною різницею в тому, що до мітки *PROC* потрібно додати мову програмування, на якій дана функція буде викликана: *Ім’я\_функції PROC Мова\_програмування.*

Щоб оголосити дану функцію на мові програмування, наприклад, С, потрібно написати прототип функції: *extern “Мова\_програмування” тип\_значення\_повернення Ім'я\_функції(аргументи\_функції, …);.*

Виклик функції здійснюється наступним чином: *Ім’я\_функції (аргументи\_функцій, …);.*

Перед запуском програми, потрібно підготувати проект для виконання MASM коду, для цього потрібно обрати MASM в налаштуваннях збірки, а також вибрати мову MASM для файлу з розширенням *.asm.*

***Завдання***

Розробити в IDE Microsoft Visual Studio програму мовою С/С++, яка обчислює за формулою та виводить на екран значення X (розрахункову формулу та значення змінних див. у табл. 4.1 відповідно до варіанта).

Вхідні дані оголости цілими змінними та ввести з клавіатури засобами С/С++. Результат вивести на екран засобами С/С++.

Розрахунок за формулою в С/С++-програмі реалізувати за допомогою:

а) зовнішньої процедури на MASM з передачою в неї змінних в якості параметрів;

б) асемблерної вставки на MASM.

Команди програми супроводити докладними коментарями.



де *a*=100; *b*=3;*c*=2; *d*=4

***Розв’язок:***

**Б) Ассемблена вставка MASM**

Програма Source.cpp

#include <iostream>

using namespace std;

#define m 95

int main() {

    short A, B, C1, D, X;

    cout << "Enter a="; cin >> A;

    cout << "Enter b="; cin >> B;

    cout << "Enter c="; cin >> C1;

    cout << "Enter d="; cin >> D;

    \_\_asm {

        xor eax, eax                            //; ax = 0

        mov      ax, B                          //; ax = b

        mul      ax                             //; ax = b ^ 2

        mov      cx, ax                         //; cx = ax = b ^ 2

                                                //

        xor eax, eax                            //; ax = 0

        mov      ax, C1                         //; ax = c

        mul      ax                             //; ax = c ^ 2

        mul      cx                             //; ax = c ^ 2 \* b ^ 2

        sub      ax, 23d                        //; ax = (c ^ 2 \* b ^ 2) - 23

        mov      cx, ax                         //; cx = ax = (c ^ 2 \* b ^ 2) - 23

                                                //

        xor eax, eax                            //; ax = 0

        mov      ax, D                          //; ax = d

        mul      ax; ax = d ^ 2                 //

        sub      ax, B                          //; ax = d ^ 2 - b

        mov      bx, A                          //; bx = a

        mul      bx                             //; dx:ax = (d ^ 2 - b) \* a

        div      cx                             //; ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23

        mov      cx, ax                         //; cx = ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23 --->cx

                                                //

        xor eax, eax                            //; ax = 0

        mov      bx, B                          //; bx = b

        add      bx, cx                         //; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23)

        sub      bx, 95d                        //; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23) - 95

        mov[X], bx

    }

    cout << "Result: " << X << endl;

    return 0;

}

***Результати роботи****:* (див. рис. 4.1)

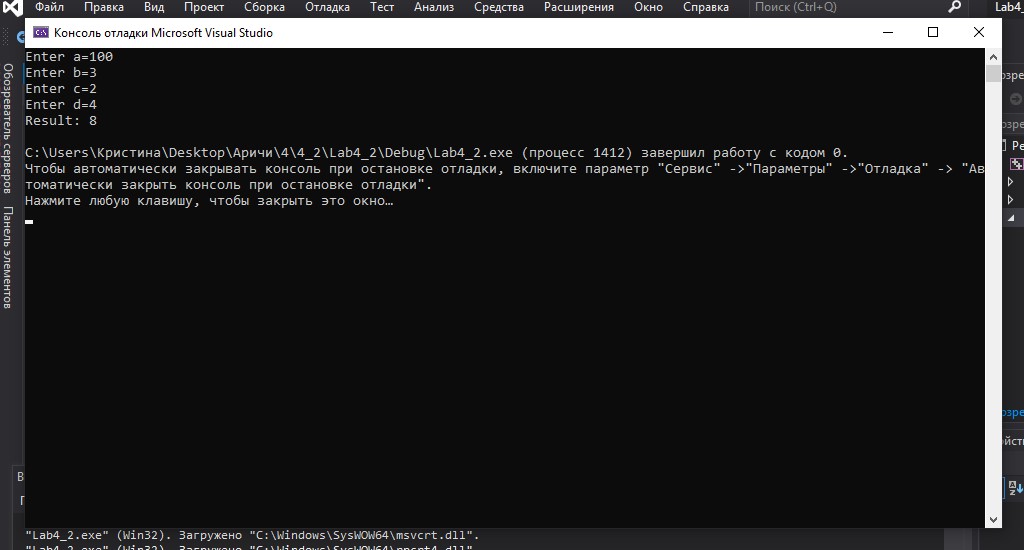


Рис. 4.1 – Результати роботи програми методом ассемблерної вставки

**А) Зовнішня процедура**

Програма main.cpp

#include <iostream>

using namespace std;

extern "C" int Expression(int, int, int, int);

int main()

{

    int a, b, c, d, x;

    cout << "Enter a="; cin >> a;

    cout << "Enter b="; cin >> b;

    cout << "Enter c="; cin >> c;

    cout << "Enter d="; cin >> d;

    cout << "Result: " << Expression(a, b, c, d) << endl;

    system("pause");

    return 0;

}

Програма Source.asm

.586

.MODEL FLAT, C

.DATA

m EQU 95

.CODE

Expression PROC C

push ebp

mov ebp, esp

xor eax, eax ; ax = 0

mov ax, [ebp+12] ; ax = b

mul ax ; ax = b ^ 2

mov cx, ax ; cx = ax = b ^ 2

xor eax, eax ; ax = 0

mov ax, [ebp+16] ; ax = c

mul ax ; ax = c ^ 2

mul cx ; ax = c ^ 2 \* b ^ 2

sub ax, 23d ; ax = (c ^ 2 \* b ^ 2) - 23

mov cx, ax ; cx = ax = (c ^ 2 \* b ^ 2) - 23

xor eax, eax ; ax = 0

mov ax, [ebp+20] ; ax = d

mul ax ; ax = d ^ 2

sub ax, [ebp+12] ; ax = d ^ 2 - b

mov bx, [ebp+8] ; bx = a

mul bx ; dx:ax = (d ^ 2 - b) \* a

div cx ; ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23

mov cx, ax ; cx = ax = (d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23 ---> cx

xor eax, eax ; ax = 0

mov bx, [ebp+12] ; bx = b

add bx, cx ; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23)

sub bx, 95d ; bx = b + ((d ^ 2 - b) \* a / (c ^ 2 \* b ^ 2) - 23) - 95

mov ax, bx

mov esp ,ebp

pop ebp

RET

Expression Endp

END

***Результати роботи****:* (див. рис. 4.2)

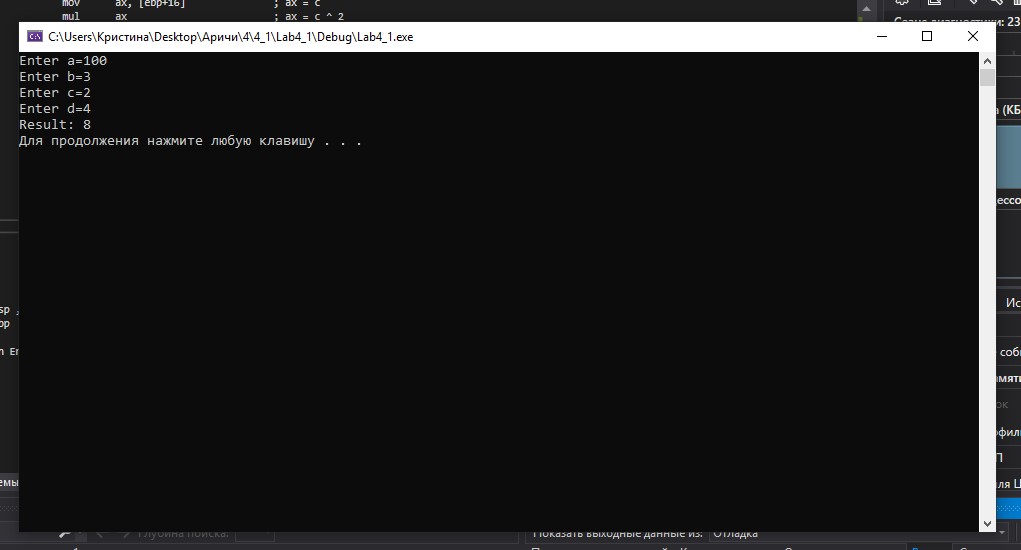


Рис. 4.2 – Результати роботи програми методом зовнішньої процедури

***Висновок.***

Познайомились з методом оптимізації програм, використовуючи вставки на мовах низького рівня. Набули навичок застосування зовнішніх процедур та вставок на асемблері у програмах на С/С++.

#### Лабораторна робота №5.

**Програмування математичного співпроцесора в програмах на Assembler**

***Мета роботи*:** вивчення основних команд математичного співпроцесора; набуття навичок застосування цих команд для програмування на MASM.

***Теоретичні основи*.**

Математичний співпроцесор — співпроцесор для розширення системи команд центрального процесора, що забезпечується функціональністю модуля операцій з плаваючою комою для процесорів, які не мають відповідного інтегрованого модуля. Співпроцесор не є повноцінним процесором, оскільки не вміє виконувати багатьох операцій (наприклад, не вміє працювати з програмою і обчислювати адреси пам'яті), він є всього лише додатком до центрального процесора.

Модуль операцій з плаваючою комою (або з плаваючою крапкою) — частина процесора для виконання широкого спектру математичних операцій над числами.

Простим «цілочисловим» процесорам для роботи з дійсними числами і математичними операціями потрібні відповідні процедури підтримки та час для їх виконання. Модуль операцій з плаваючою комою підтримує роботу з ними на рівні примітивів — завантаження, вивантаження дійсного числа (в спеціалізовані регістри) або математична операція над ними виконується однією командою, за рахунок цього досягається значне прискорення таких операцій.

***Команди співпроцесора.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Команда** | **Операнди** | **Пояснення** | **Опис** |
| Команди передачі даних дійсного типу | | | |
| FLD | src | TOPSWR-=1; ST(0)=src; | Загрузка операнда на вершину стеку |
| FST | dst | dst=ST(0); | Збереження вершини стеку в пам’ять |
| Арифметичні команди дійсного типу | | | |
| FADD | dst, src | dst = dst + src; | Дійсне додавання |
| FSUB | dst, src | dst = dst - src; | Дійсне віднімання |
| FMUL | dst, src | dst = dst \* src; | Дійсне множення |
| FDIV | dst, src | dst = dst / src; | Дійсне ділення |

***Завдання.***

Розробити в IDE Microsoft Visual Studio програму мовою С/С++, яка обчислює за формулою та виводить на екран значення X (розрахункову формулу та значення змінних див. у табл. 5.1 відповідно до варіанта).

Вхідні дані оголосити дійсними змінними та ввести з клавіатури засобами С/С++. Результат вивести на екран засобами С/С++.

Розрахунок за формулою в С/С++-програмі реалізувати за допомогою:

а) зовнішньої процедури на MASM з передачою в неї змінних в якості параметрів;

б) асемблерної вставки на MASM.

Команди програми супроводити докладними коментарями.



де *a = -5; b = 10; c = 5; d = 22;*

***Розв’язок:***

**А) Зовнішня процедура**

Програма main.cpp

#include <iostream>

using namespace std;

const double k = 95, m = 23;

extern "C" double Expression(double, double, double, double, double, double);

int main()

{

    double a, b, c, d;

    cout << "Enter a: "; cin >> a;

    cout << "Enter b: "; cin >> b;

    cout << "Enter c: "; cin >> c;

    cout << "Enter d: "; cin >> d;

    cout << "Result: " << Expression(a, b, c, d, k, m) << endl;

    system("pause");

    return 0;

}

Програма Source.asm

.586

.MODEL FLAT, C

.DATA

.CODE

Expression PROC C

push ebp

mov ebp, esp

finit

fld qword ptr[ebp+40] ;st(0) = 95

fld qword ptr[ebp+16] ;st(0) = b

fsub st(0), st(1) ;st(0) = b - 95

fld qword ptr[ebp+16] ;st(0) = b

fmul st(0), st(0) ;st(0) = b ^ 2

fld qword ptr[ebp+48] ;st(0) = 23

fld qword ptr[ebp+24] ;st(0) = c

fmul st(0), st(0) ;st(0) = c ^ 2

fmul st(0), st(2) ;st(0) = c ^ 2 \* b ^ 2

fsub st(0), st(1) ;st(0) = (c ^ 2 \* b ^ 2) - 23

fld qword ptr[ebp+16] ;st(0) = b

fld qword ptr[ebp+32] ;st(0) = d

fmul st(0), st(0) ;st(0) = d ^ 2

fsub st(0), st(1) ;st(0) = d ^ 2 - b

fld qword ptr[ebp+8] ;st(0) = a

fmul st(0), st(1) ;st(0) = a \* (d ^ 2 - b)

fdiv st(0), st(3) ;st(0) = a \* (d ^ 2 - b) / (c ^ 2 \* b ^ 2) - 23

fadd st(0), st(6) ;st(0) = (a \* (d ^ 2 - b) / (c ^ 2 \* b ^ 2) - 23) + (b - 95)

mov esp, ebp

pop ebp

RET

Expression Endp

END

***Результати роботи****:* (див. рис. 5.1 та 5.2 )

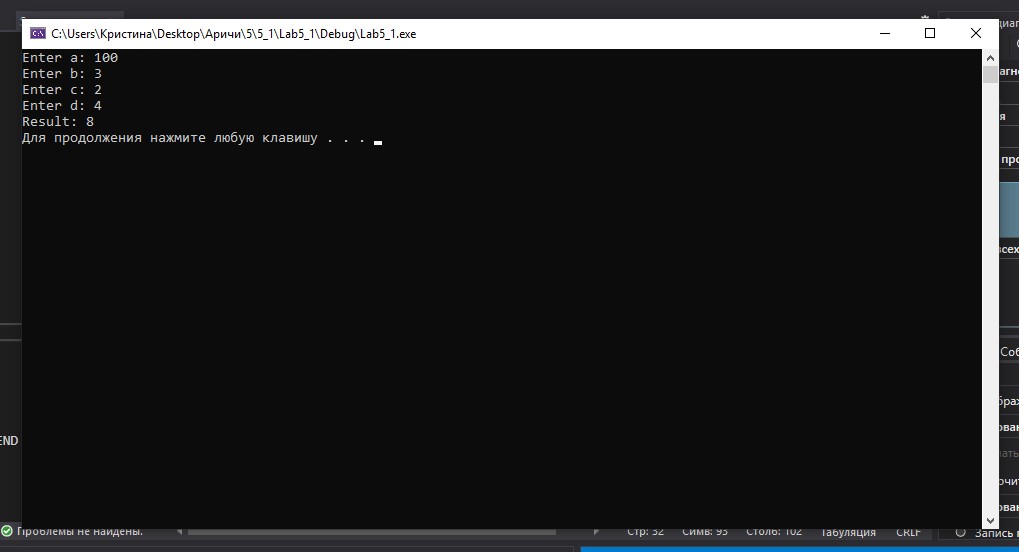


Рис. 5.1 – Результати роботи програм(з цілими числами)

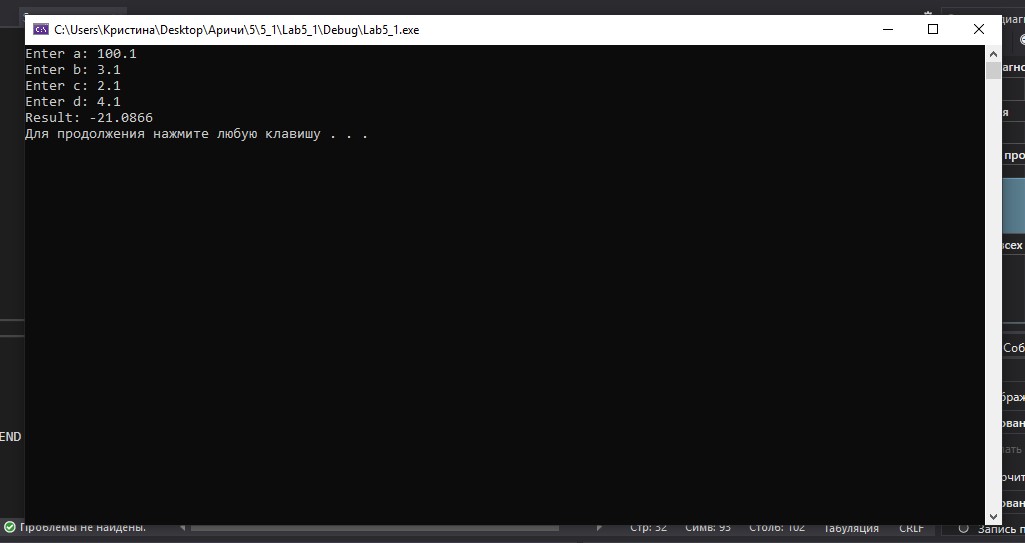


Рис. 5.2 – Результати роботи програм(з числами з плаваючою крапкою)

**Б) Ассемблена вставка MASM**

Програма main.cpp

#include <iostream>

using namespace std;

int main() {

    double A, B, C1, D, M = 95, K = 23, X;

    cout << "Enter a="; cin >> A;

    cout << "Enter b="; cin >> B;

    cout << "Enter c="; cin >> C1;

    cout << "Enter d="; cin >> D;

    \_\_asm {

        finit

        fld B                       // st(0) = b

        fmul st(0), st(0)           // st(0) = b ^ 2

        fld K                       // st(0) = 23

        fld C1                      // st(0) = c

        fmul st(0), st(0)           // st(0) = c ^ 2

        fmul st(0), st(2)           // st(0) = c ^ 2 \* b ^ 2

        fsub st(0), st(1)           // st(0) = (c ^ 2 \* b ^ 2) - 23

        fld D                       // st(0) = d

        fmul st(0), st(0)           // st(0) = d ^ 2

        fsub B                      // st(0) = d ^ 2 - b

        fld A                       // st(0) = a

        fmul st(0), st(1)           // st(0) = a \* (d ^ 2 - b)

        fdiv st(0), st(2)           // st(0) = a \* (d ^ 2 - b) / (c ^ 2 \* b ^ 2) - 23

        fld B                       // st(0) = b

        fsub M                      // st(0) = b - 95

        fadd st(0), st(1)           // st(0) = (a \* (d ^ 2 - b) / (c ^ 2 \* b ^ 2) - 23) + (b - 95)

        fst X

    }

    cout << "Result: " << X << endl;

    system("pause");

    return 0;

}

***Результати роботи****:* (див. рис. 5.3 та 5.4)

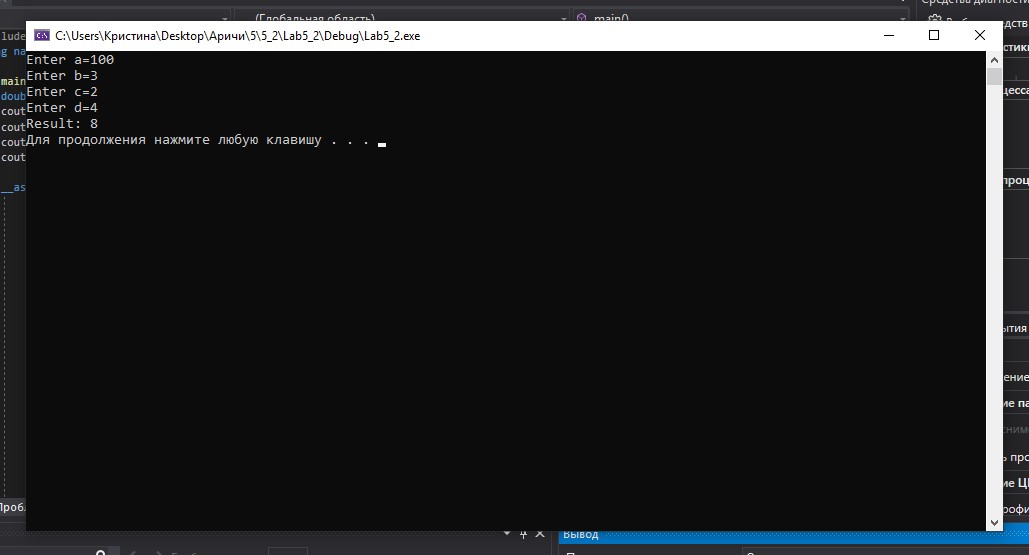


Рис. 5.3 – Результати роботи програм(з цілими числами)

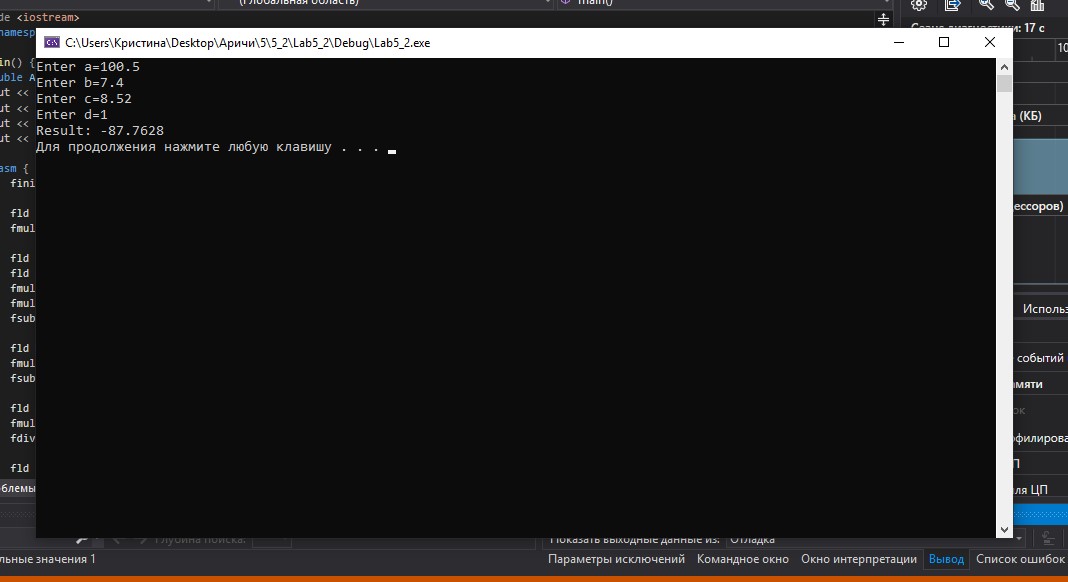


Рис. 5.4 – Результати роботи програм(з числами з плаваючою крапкою)

**Висновок.**

Познайомились з різними сімействами співпроцесорів. Набули навичок застосування команд математичного співпроцесора при програмуванні на МАSM. Навчились створювати програми на С++ використовуючи зовнішні функції на МАСМ.