ЛАБОРАТОРНА РОБОТА №1.
ЗМІШАНЕ ПРОГРАМУВАННЯ МОВАМИ C/C++ ТА АСЕМБЛЕРА (4 год)

Мета: оволодіти навиками створення програм, частини яких написані різними мовами програмування. Засвоїти правила взаємодії між програмними модулями різних мов програмування.

ТЕОРЕТИЧНІ ВІДОМОСТІ

У компіляторі *Microsoft Visual C/C++* прийнято декілька різних угод про використання зовнішніх імен та виклики функцій. Для налагодження програм та прив'язування коду до процедур мовою асемблера необхідно знати ці підходи.

Особливості інтерфейсу взаємодії для платформи *x86*. На платформі *x86* всі аргументи функції передаються через стек як 32 бітні дані. Значення, що повертається, також розширюється до 32 біт і повертається в регістрі *EAX* або у регістрі *ST(0)*, якщо результат дійсне число. Параметри розміщуються в стеку справа наліво. Компілятор створює код прологу та епілогу для збереження та відновлення регістрів *ESI*, *EDI*, *EBX* та *EBP*, якщо вони використовуються у функції.

Компілятор *Microsoft Visual C/C++* підтримує такі угоди про виклики:

_cdecl — це угода про стандартні виклики для програм C та C++. Так як стек очищає викликаюча функція, то є можливість реалізувати функції з змінною кількістю параметрів. У наступній таблиці показано реалізацію цієї угоди про виклики.

Таблиця 1.1

Реалізація угоди про виклики *_cdecl*

Елемент	*Реалізація*
Передача аргументів	Через стек справа наліво
Очистка стеку	Викликаюча функція
Оформлення імен	Перед іменами ставиться символ підкреслення _

_stdcall – це угода про виклики, що використовується для виклику функцій API Win32. Для функцій, що використовують цю угоду про виклики, необхідний прототип. У наступній таблиці показано реалізацію цієї угоди про виклики.

Таблиця 1.2

Реалізація угоди про виклики *_stdcall*

Елемент	*Реалізація*
Передача аргументів	Через стек справа наліво
Очистка стеку	Викликана функція
Оформлення імен	Перед іменами ставиться символ підкреслення _. За ім'ям слідує знак @, за яким слідує число байт (у десятковому форматі) з списку аргументів. Тому функція, оголошена як int func(int a, double b) декорується так: _func@12

_fastcall – ця угода про виклики вказує на те, що аргументи для функцій повинні передаватися в регістрах, коли це можливо. Ця угода про виклики застосовується лише до архітектури *x86*. У наступному списку показано реалізацію цієї угоди про виклики. У різних версіях компілятора можуть використовуватись інші регістри процесора для збереження параметрів!

Таблиця 1.3

Реалізація угоди про виклики *_fastcall*

Елемент	*Реалізація*
Передача аргументів	Перші два значення розміром *DWORD* або меншим, знайдені у списку аргументів зліва направо, передаються в регістрах *ECX* та *EDX*; решта аргументів передаються в стек справа наліво.
Очистка стеку	Викликана функція
Оформлення імен	Перед іменами ставиться знак *@,* після імені ставиться знак @ за яким слідує кількість байтів (у десятковій системі числення) у списку параметрів

Особливості написання коду прологу та епілогу функції. Функції, написані на мові асемблера, можуть звертатися до параметрів, переданих через стек, з використанням регістру *EBP*.

У цьому прикладі показаний стандартний код прологу, який може бути присутнім у 32-розрядній функції:

```
push    ebp             ; Save ebp
mov     ebp, esp        ; Set stack frame pointer
sub     esp, localbytes ; Allocate space for locals
push    <registers>     ; Save registers
```

Значення localbytes показує число байт, які потрібні в стеку для локальних змінних, а значення <registers> - це заповнювач, що представляє список регістрів,

які необхідно зберегти в стеку. Після зберігання регістрів можна розмістити в стеку всі інші дані. Нижче наведено відповідний код епілогу.

```
pop     <registers>     ; Restore registers
mov     esp, ebp        ; Restore stack pointer
pop     ebp             ; Restore ebp
ret                     ; Return from function
```

Стек завжди розширюється у напрямку донизу (від старших адрес пам'яті до молодших). Вказівник бази *EBP* вказує на розміщене у стеку попереднє значення *EBP*. Область локальних змінних починається з *EBP-4*. Область параметрів починається з *EBP+8*. У *EBP+4* зберігається адреса повернення. Для доступу до локальних змінних, а також для параметрів функції необхідно обчислити зсув від *EBP* шляхом віднімання (чи додавання для параметрів функції) відповідного значення від *EBP*:

[EBP + 12] – другий параметр функції
[EBP + 08] – перший параметр функції
[EBP + 04] – адреса повернення
[EBP] – значення EBP перед прологом
[EBP – 04] – перша локальна змінна
[EBP – 08] – друга локальна змінна

Наприклад виклик функції Test(i, j, 1); за угодою *_cdecl* скомпілюється в такі інструкції:

```
mov  eax, 1
push  eax
push  dword ptr _j
push  dword ptr _i
call  _Test
add esp,12
```

де видно, що правий параметр (значення 1), заноситься в стек першим, потім туди заноситься параметр *j* та, нарешті, *i*.

При поверненні з функції занесені в стек параметри усе ще знаходяться там, але вони більше не використовуються. Тому після кожного виклику функції за угодою *_cdecl* необхідно прокрутити вказівник стеку назад у відповідності зі значенням, що він мав перед занесенням у стек параметрів. У попередньому прикладі три параметри (по чотири байти кожен) займають у стеку разом 12 байт, тому компілятор додає значення 12 до вказівника стеку, щоб відкинути параметри після звертання до функції Test.

На рисунку 1.1 показано, як виглядає стек перед виконанням першої інструкції у функції Test.

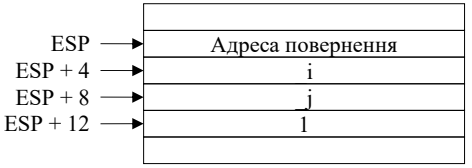

Рис. 1.1. Стан стеку перед виконанням першої інструкції функції Test.

На рисунку 1.2 показано, як виглядає стек після виконання прологу функції Test.

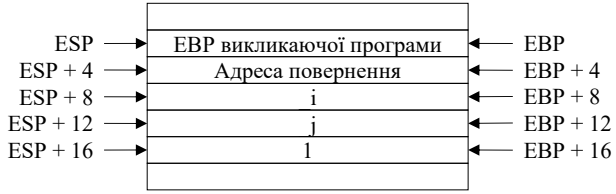

Рис. 1.2. Стан стеку після виконання прологу функції Test.

Особливості інтерфейсу взаємодії для платформи *x64*. На платформі *x64* є тільки одна угода про виклики (модифікований варіант угоди *_fastcall*), яка використовується по замовчуванню і реалізована наступним чином:

Передача параметрів – перші чотири параметри функції передаються через регістри. Параметри, що залишилися, передаються в стек у порядку справа наліво. Цілочисельні параметри в перших чотирьох позиціях передаються в порядку зліва направо у регістри *RCX*, *RDX*, *R8* і *R9*, відповідно. П'ятий та наступні аргументи передаються у стек, як описано вище. Всі цілі аргументи в регістрах вирівнюються по правому краю, тому об'єкт, що викликається, може ігнорувати верхні біти регістру і отримати доступ тільки до тієї частини регістра, яка необхідна. Будь-які аргументи з плаваючою комою та подвійною точністю в перших чотирьох параметрах передаються у регістри *XMM0–XMM3* (залежно від позиції).

Повернення результату – цілочисельний результат повертається за допомогою регістру *RAX*. Числа з плаваючою комою повертаються за допомогою регістру *XMM0*.

Регістри *RAX, RCX, RDX, R8, R9, R10, R11* і *XMM0–XMM5* розглядаються як такі, що змінюються. Регістри *RBX, RBP, RDI, RSI, RSP, R12, R13, R14, R15* і

XMM6-XMM15 є незмінними. Вони мають бути збережені та відновлені за допомогою функції, яка їх використовує.

Функція, яка буде викликати інші функції, обов'язково повинна в області стеку зарезервувати місце під перші чотири параметри функції (32 байти), яка буде викликатися. Це місце використовується у функціях для копіювання значень параметрів з регістрів, що забезпечує звертання до параметрів «по-старому» (*[RBP + xx]*). Очищення стеку лежить на викликаючій функції.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке змішане програмування?
2. Як передаються параметри функціям, написаним під 32-х розрядну платформу?
3. Як передаються параметри функціям, написаним під 64-х розрядну платформу?
4. Як передаються результати з функцій?
5. Хто має очищати стек після виклику функції, що написана згідно угоди *_cdecl*?
6. Хто має очищати стек після виклику функції, що написана згідно угоди *_stdcall*?
7. Хто має резервувати місце під параметри і очищати стек для виклику функції, яка написана під 64-х розрядну платформу?
8. Особливості взаємодії С-Асемблер-С.

ЛІТЕРАТУРА

1. Calling Conventions [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-gb/cpp/cpp/calling-conventions?view=msvc-170.
2. x64 calling convention [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-gb/cpp/build/x64-calling-convention?view=msvc-170.
3. Рисований О.М. Системне програмування: підручник для студентів напрямку "Комп'ютерна інженерія" вищих навчальних закладів в 2-х томах. Том 1. – Видання четверте: виправлено та доповнено – Х.: "Слово", 2015. – 576 с.

ЗАВДАННЯ

1. Створити перший програмний проект **для 32-х розрядної платформи**, який реалізує схему викликів **C-ASM-C** та здійснює обчислення заданого виразу, згідно варіанту (таблиця 1.4). Проект повинний складатися з двох модулів, передача параметрів між якими здійснюється через стек. Константа передається через спільну пам'ять, як глобальні дані. Множення/ділення рекомендовано робити через команди зсувів.

   Основний модуль – створюється мовою C/C++. Він повинен забезпечувати:

   - ввід даних з клавіатури;
   - обчислення виразу засобами мови C/C++;
   - вивід на екран результатів обчислення виразу.
   - виклик асемблерної процедури обчислення виразу.

   Модуль безпосередніх обчислень (асемблерний модуль) – здійснює обчислення виразу засобами асемблера і реалізує вивід на екран результату обчислення шляхом виклику стандартної функції мови C/C++ `printf()`, або її аналога.

2. Створити другий програмний проект **для 64-х розрядної платформи**, який також реалізує схему викликів **C-ASM-C** та здійснює обчислення заданого виразу, згідно варіанту. Проект повинна складатися з двох модулів, передача параметрів між якими повинна відбуватися, згідно угод для платформи x64. Константа передається через спільну пам'ять, як глобальна змінна. Множення/ділення рекомендовано робити через команди зсувів.

   Призначення програмних модулів таке ж, як в першому програмному проекті.

3. Відлагодити та протестувати обидва програмні проекти. Результати роботи програм продемонструвати викладачу.
4. Скласти звіт про виконану роботу з приведенням текстів програм та коментарів до неї, а також результатів її роботи.
5. Дати відповідь на контрольні запитання викладача.

Таблиця 1.4.

Варіанти завдань

№	Вираз	K
1.	$X=A_2+C_1-D_2/2+K$	1254021
2.	$X=A_4+C_2+D_1*4-K$	202
3.	$X=K-B_2*8+C_2-E_1$	37788663
4.	$X=A_4+C_1-D_4/8+K$	45694
5.	$X=B_4-A_2*2-E_2+K$	505
6.	$X=K+B_2/4-D_2*4-E_1$	6DD02316
7.	$X=A_4/2-4*(D_1+E_2-K)$	717
8.	$X=A_4-B_2+K-D_2/2+8*B_2$	88
9.	$X=4*B_2-C_2+D_4/4$	29
10.	$X=A_4-B_4/2+K+E_2*4$	2310
11.	$X=(A_4-B_2-K)*2+E_4/4$	311
12.	$X=K+B_4/2-4*F_2-E_1$	7055E0AC
13.	$X=A_2/2+8*(D_1+E_2-K)$	2513
14.	$X=A_4-B_1-K-D_2/2+4*B_1$	614
15.	$X=A_4+(4*C_2)-D_4/2+K$	4569600F
16.	$X=A_4/4+C_2-D_1*2+K$	616
17.	$X=A_4-K+C_4/2-E_1*8$	1017
18.	$X=4*(B_2-C_1)+D_2/4+K$	56987018
19.	$X=A_2*4+C_1-D_4/2+K$	4019
20.	$X=K+B_4/4-D_1*2-E_2$	18932020
21.	$X=A_4/8+2*(D_2-E_1+K)$	21
22.	$X=K-B_1-C_1-D_2/2+4*B_1$	45781022
23.	$X=A_2*8-C_1+D_4/2+K$	7AA02023
24.	$X=K-B_2/2+D_4+E_2*4$	74569024
25.	$X=(K-B_2-C_1)*2+E_4/4$	2B05025
26.	$X=A_2+K+C_2/2-E_1*8$	6C26
27.	$X=A_2*4+(K-E_1*4)$	A77627
28.	$X=K+B_4/2+D_2-E_2/4$	3FF28
29.	$X=K-B_1*4+D_2-F_2/2$	12A0C029
30.	$X=K+B_4-D_2/2+E_1*4$	25630

де: A, B, C, D, E, F - знакові цілі числа, довжиною в байтах, згідно з індексом, значення константи K подано у 16-ій системі числення.

ПОРЯДОК ВИКОНАННЯ

1. Запустити середовище розробки *Microsoft Visual Studio 2022* (або *Microsoft Visual Studio 2019*).

2. Створити новий проект:

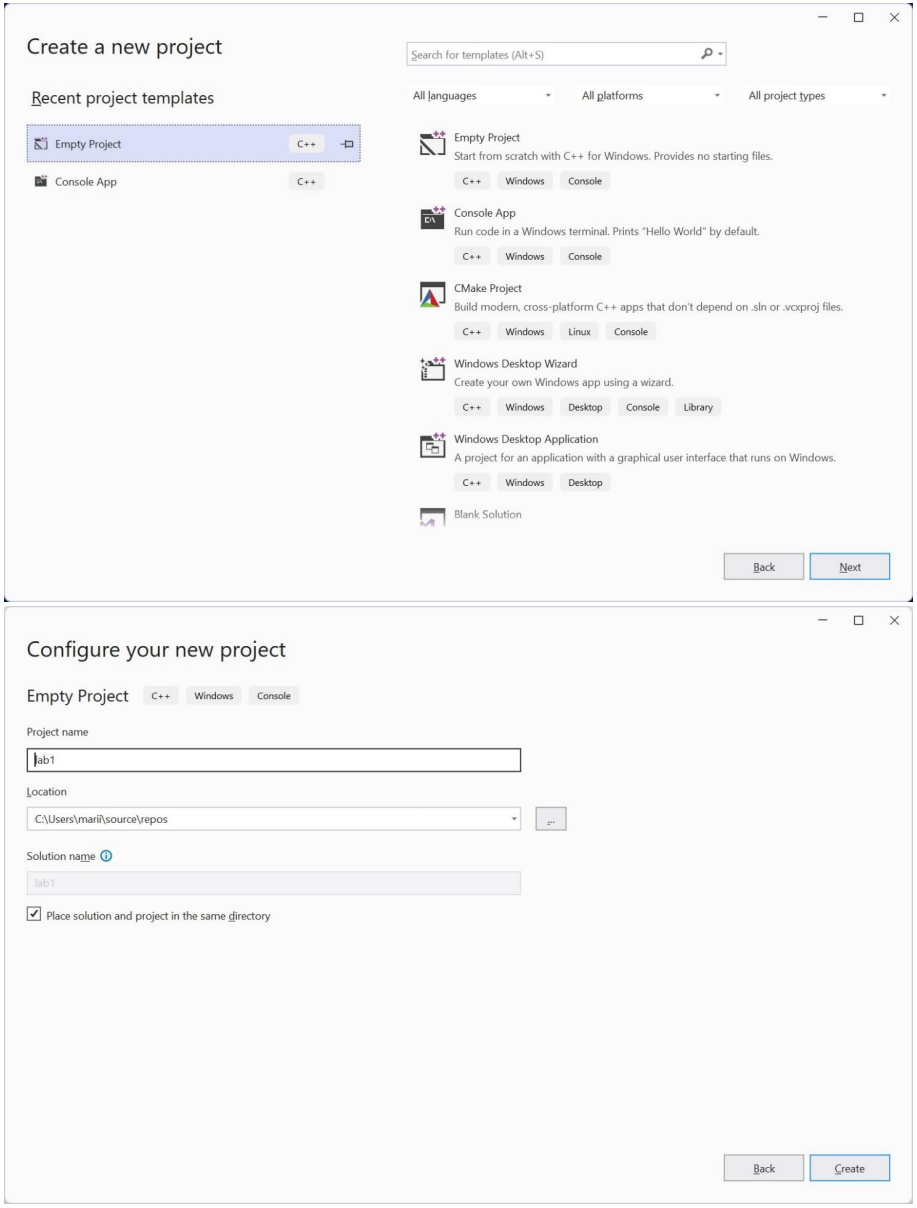

Рис. 1.3. Створення нового проекту в середовищі Microsoft Visual Studio 2022.

3. У залежностях збірки проекту (натискаємо правою кнопкою на ім'я проекту, далі вибираємо ***Build Dependencies/Build Customizations***) ставимо галочку навпроти *masm*:

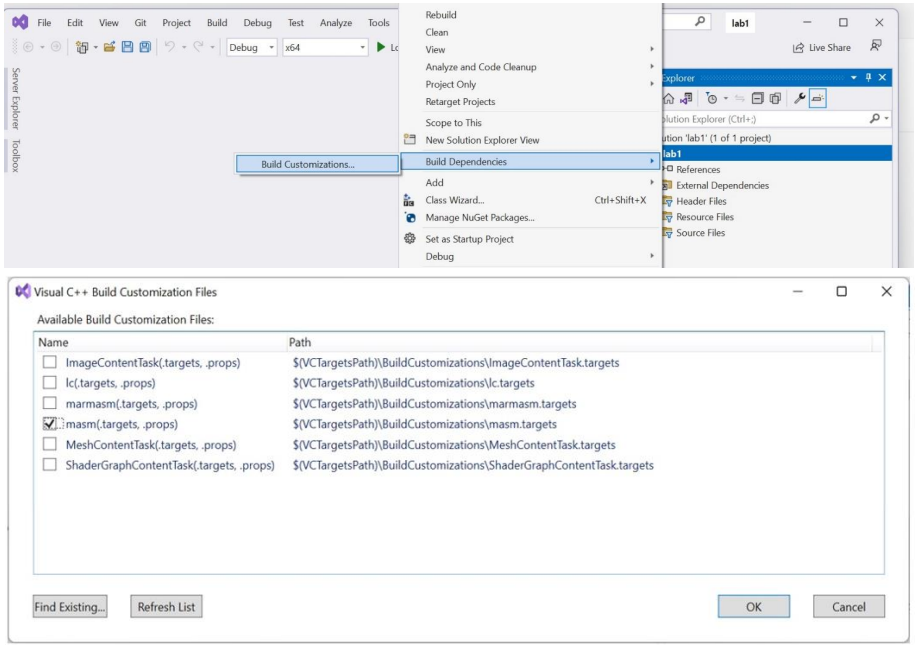

Рис. 1.4. Налаштування проекту.

4. До створеного проекту додати файли для вихідних текстів програми. Це можна зробити через меню ***Project/Add Module***. Один з файлів має мати розширення *.cpp*, а інший *.asm*:

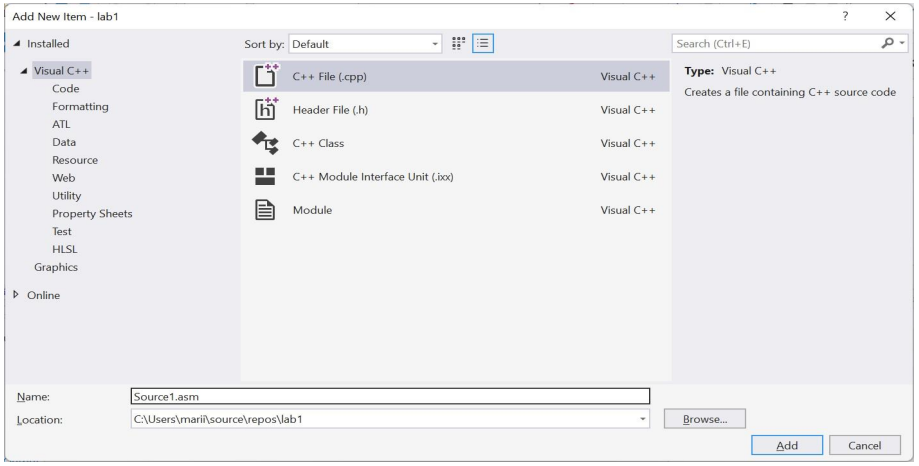

Рис. 1.5. Додавання файлів у проект.

Для файлу з розширенням *.asm* необхідно переконатися, що у його властивостях вибрано тип *Microsoft Macro Assembler*:

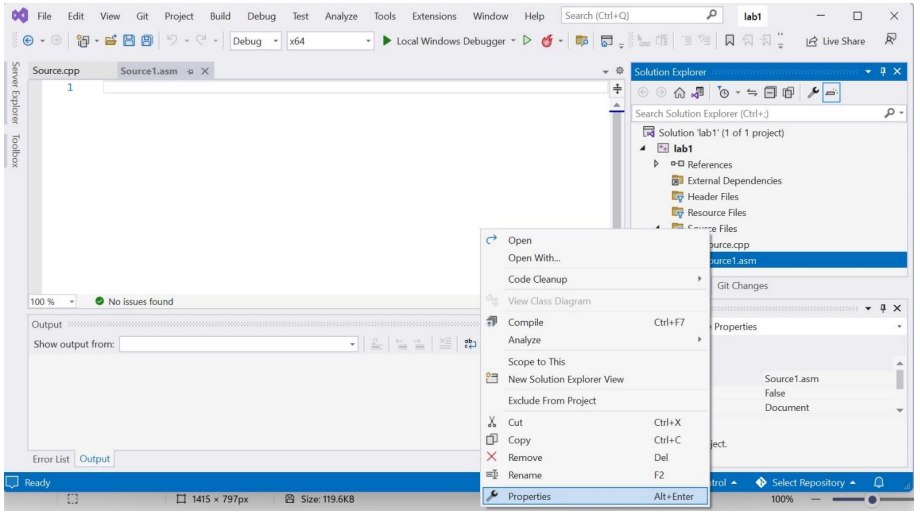

Рис. 1.6. Властивості файлу з розширенням .asm

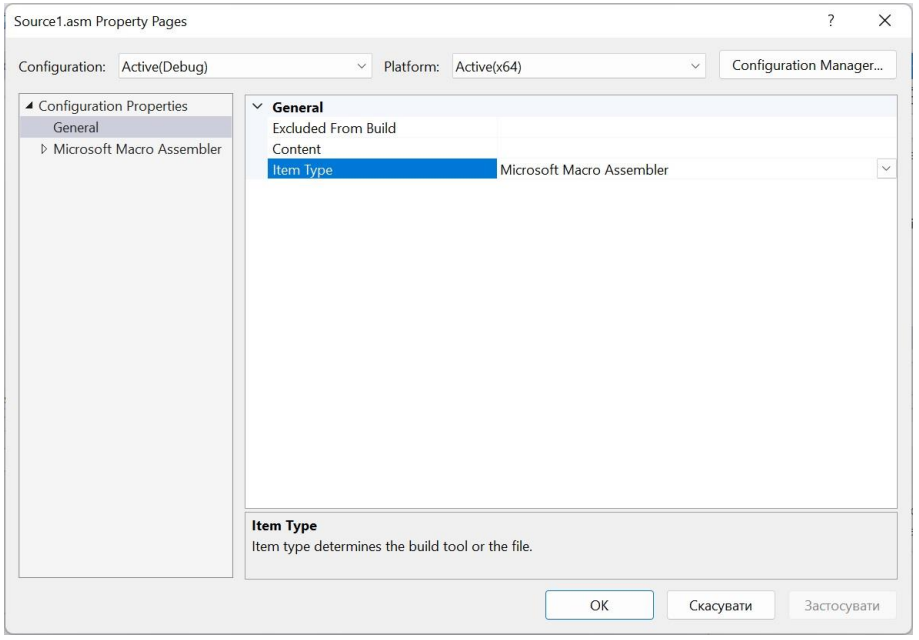

Рис. 1.7. Тип .asm файлу Microsoft Macro Assembler

5. Скопіювати наведені в методичних вказівках коди програми у відповідні файли з розширеннями *.cpp* та *.asm*.

6. Скомпілювати створений проект, використавши вкладку ***Build***. При компілюванні проекту під 32-х розрядну платформу необхідно вибрати пункт *x86*:

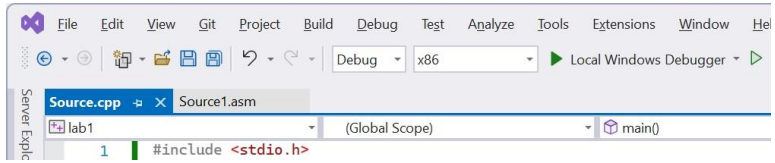

Рис. 1.8. Налаштування під 32-х розрядну платформу

При компілюванні проекту під 64-х розрядну платформу необхідно вибрати пункт *x64*:

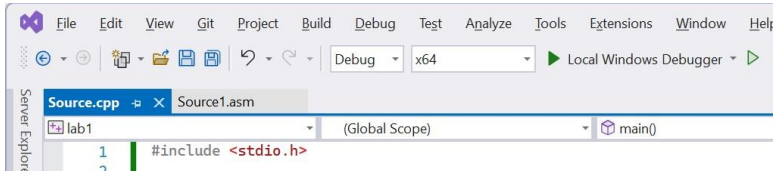

Рис. 1.9. Налаштування під 64-х розрядну платформу

7. Модифікувати тестові програми відповідно до заданого варіанту, скомпілювати та запустити на виконання.

ПРИКЛАДИ ПРОГРАМ

Програма, що виконує обчислення виразу «A + B + 100» та ілюструє взаємодію *C-ASM-C* для 32-х розрядної платформи:

main.cpp

```
#include <stdio.h>

extern "C" int calc(int, int);
extern "C" int K = 100;

int main()
{
	int a;
	int b;
	int res;
	printf("Enter numbers:\n");
	printf("A = ");
	scanf_s("%d", &a);
	printf("B = ");
	scanf_s("%d", &b);
	printf("\nA + B + 100 = %d\n", (a + b + K));
	res = calc(a, b);
	printf("\nResult of procedure calc is: %d\n", res);
	return 0;
}
```

calc.asm

```
.686
.model flat, c
 printf proto c : vararg
 EXTERN K : DWORD

.data
 msg db 'Output from  asm module is:  %d', 0
.code
calc PROC
  push ebp
  mov ebp,esp
  mov eax, dword ptr [ebp+8]
  add eax, dword ptr [ebp+12]
  add eax, K
  pop ebp
  push eax
  invoke printf, offset msg, eax
  pop eax
  ret
calc ENDP
END
```

Програма, що виконує обчислення виразу «A + B + 100» та ілюструє взаємодію *C-ASM-C* для 64-х розрядної платформи:

main.cpp

```cpp
#include <stdio.h>

extern "C" int calc(int, int);
extern "C" int K = 100;

int main()
{
	int a;
	int b;
	int res;
	printf("Enter numbers:\n");
	printf("A = ");
	scanf_s("%d", &a);
	printf("B = ");
	scanf_s("%d", &b);
	printf("\nA + B + 100 = %d\n", (a + b + K));
	res = calc(a, b);
	printf("\nResult of procedure calc is: %d\n", res);
	return 0;
}
```

calc.asm

```
printf proto c : vararg
EXTERN K : DWORD

.data
 msg db 'Output from asm module is:  %d', 0
.code
calc PROC
  sub rsp, 20h
  xor eax, eax
  add eax, ecx
  add eax, edx
  add eax, K
  mov r12d, eax
  lea rcx, msg
  mov edx, eax
  call printf
  mov eax, r12d
  add rsp, 20h
  ret
calc ENDP
END
```