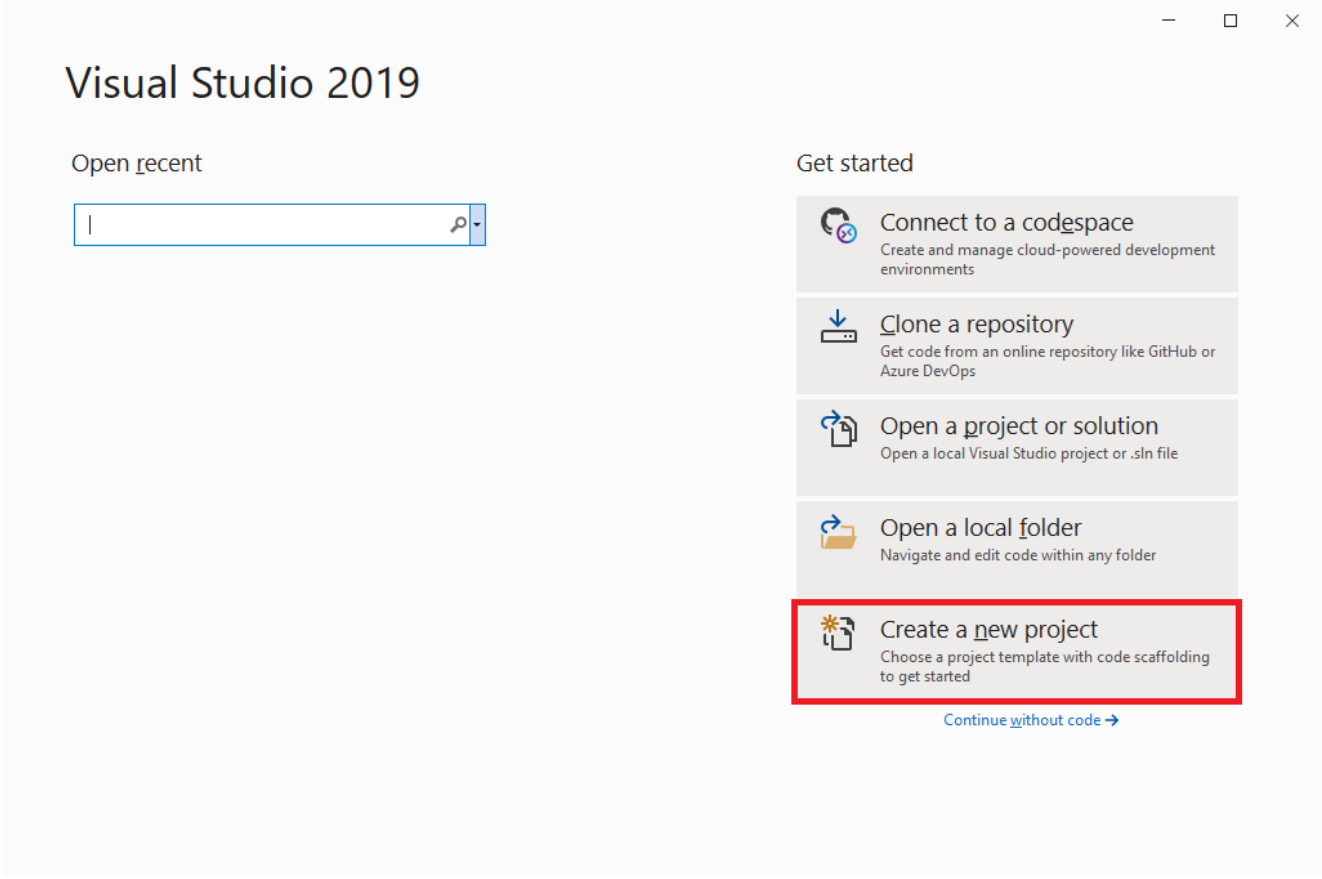
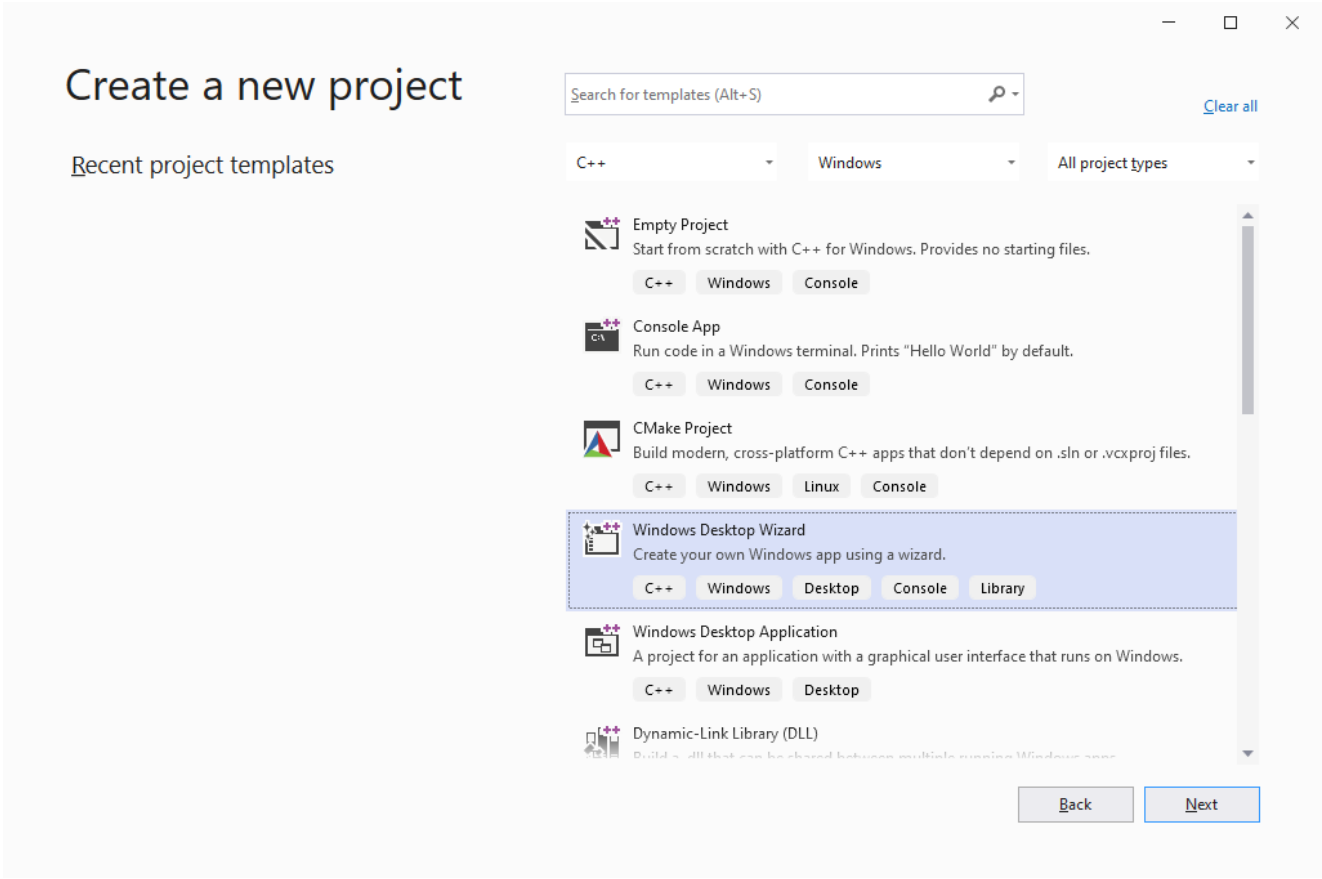
Laboratorium programowania niskopoziomowego

LAB 2 x86 - Proste podprogramy. Uruchamianie krokowe.

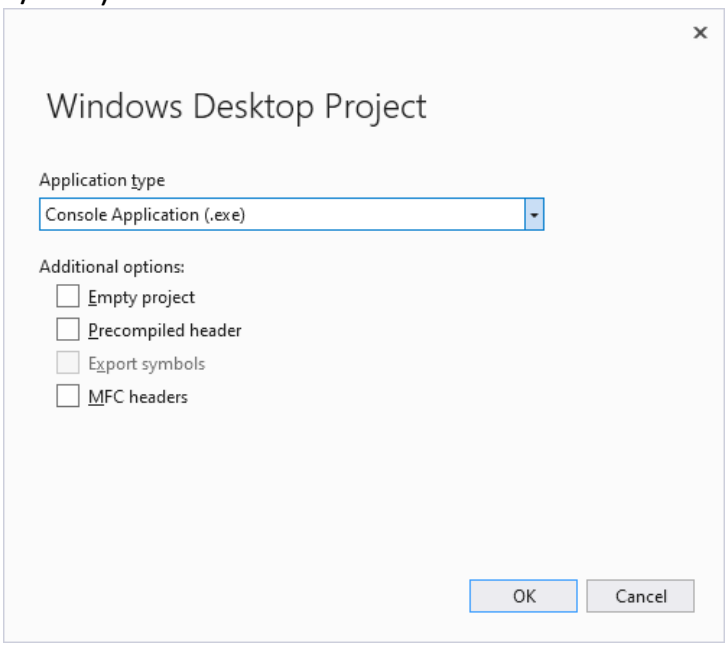

Visual Studio 2019
Open recent
Get started
Connect to a codespace
Create and manage cloud-powered development environments
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code


Create a new project
Recent project templates
Search for templates (Alt+S)
Clear all
C++
Windows
All project types
Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console
Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console
CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ Windows Linux Console
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library
Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ Windows Desktop
Dynamic-Link Library (DLL)
Back
Next

Celem ćwiczenia jest zapoznanie studenta z podstawową obsługą Visual Studio w opcji kompilacji asemblera 32 bitowego w trybie inline. W tym celu wybieramy projekt **Windows Desktop Wizard** (w poprzednich wersjach Visual Studio wybieramy Win32 Console Application) dla języka C++ (jak na załączonym screenie). (**PL:** Kreator aplikacji klasycznej systemu Windows / **UA:** / **RU:**)

Następnie wybieramy odpowiednią lokalizację i zapisujemy projekt pod konkretną nazwą.

Następnym krokiem są dodatkowe ustawienia jak na poniższym obrazku (klikamy Create) (**PL:** Aplikacja konsolowa / **UA:** / **RU:**):

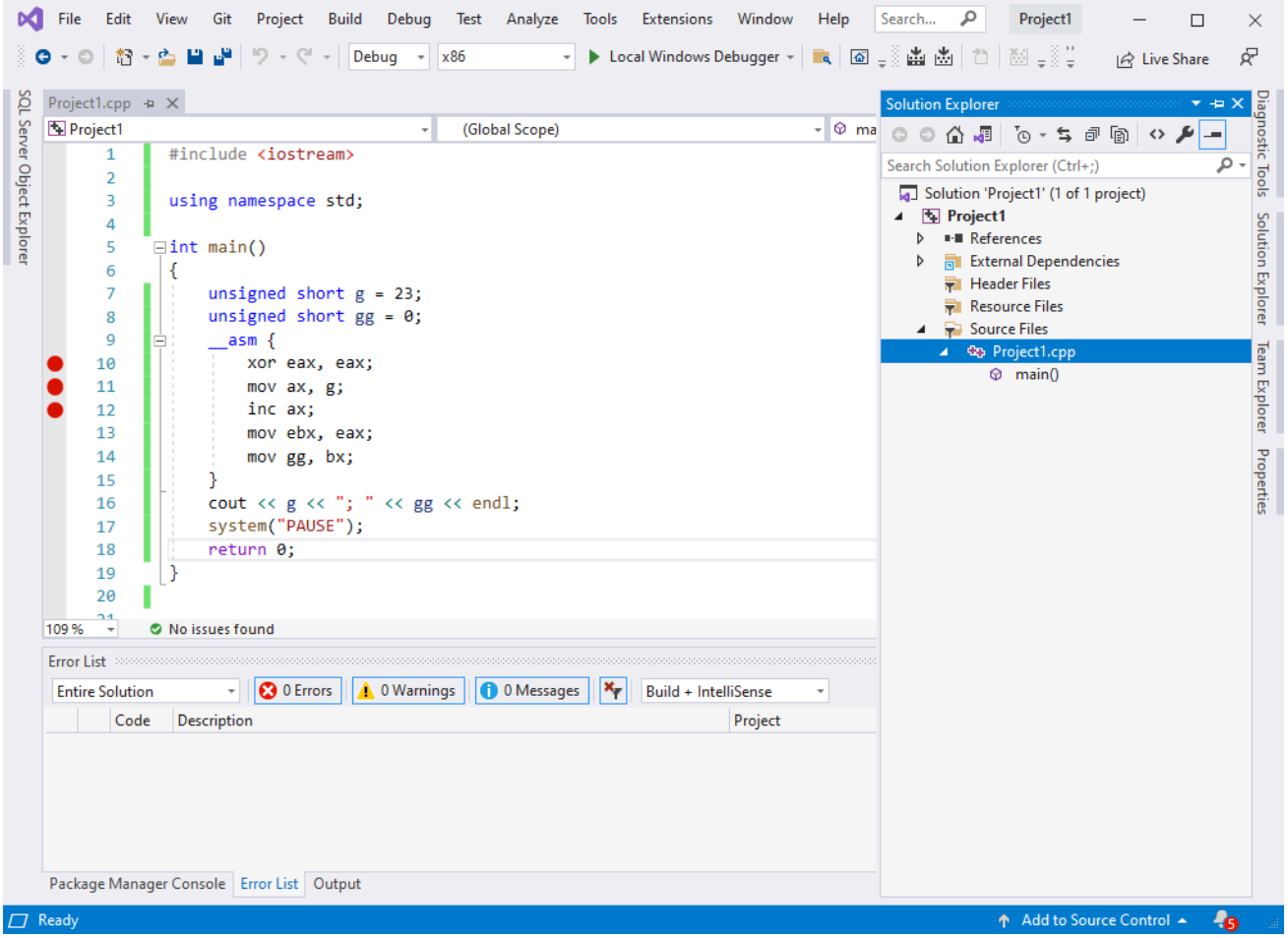


Pozostawiamy domyślne ustawienia i klikamy OK.

Teraz można wpisać kod programu.

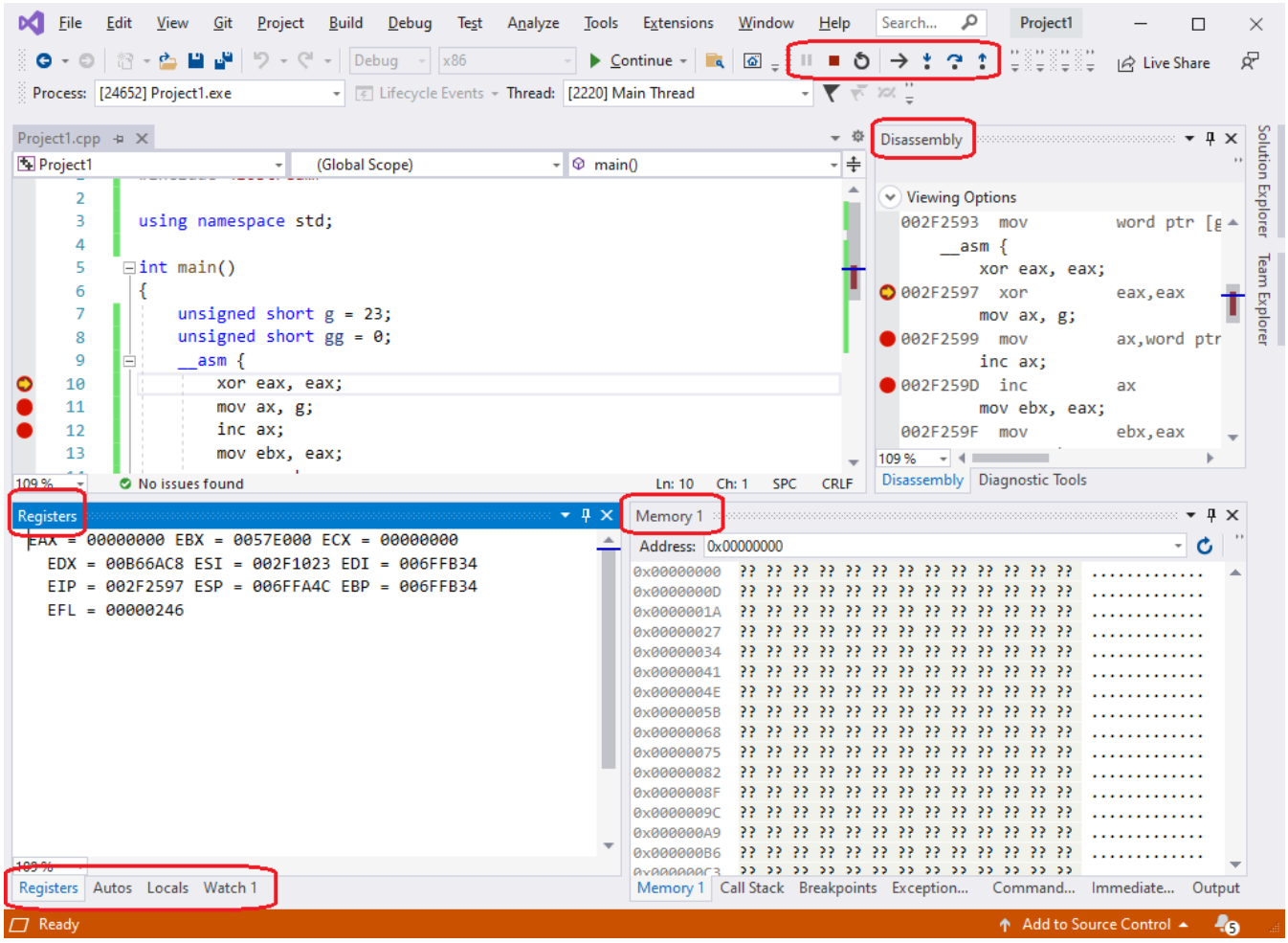
2. Drugi sposób utworzenia projektu dla asemblera.

Drugim sposobem na utworzenie projektu dla asemblera jest wybranie opcji **Empty Project** (**PL:** Pusty Projekt / **UA:** / **RU:**) i zatwierdzeniu wybranego projektu **Create** (**PL:** Utwórz / **UA:** / **RU:**). Następnie prawym przyciskiem myszy należy kliknąć na projekcie (lub jednym z przygotowanych folderów np. **Source Files** (**PL:** Pliki źródłowe / **UA:** / **RU:**)) wybrać opcję **Add** (**PL:** Dodaj / **UA:** / **RU:**) -> **New item** (**PL:** Nowy element / **UA:** / **RU:**). W wyświetlonym oknie należy wpisać nazwę pliku z rozszerzeniem (np. main.cpp).



Aby z debugować program, należy ustawić break pointy (czerwone kropki) jako punkty zatrzymań programu. Break Point określa miejsce zatrzymania programu dla danej linijki kodu przed jej wykonaniem. To oznacza, że dla załączonego obrazka, break point ustawiony na linijce 10-tej zostanie fizycznie zatrzymany przed wykonaniem tej linijki (przed wykonaniem instrukcji xor eax, eax;)

Sugestie związane z pomocnymi dodatkowymi oknami, które pomagają podczas debagowania jak i wspierają programistę w trakcie pracy krokowej.



Pomocnymi okienkami i zakładkami są (Registers, Dissasembly, Memory, Watch, Locals):

- Registers (**PL:** Rejestry / **UA:** / **RU:**):
  Wyświetla większość dostępnych rejestrów oraz ich zawartość. Dla naszych potrzeb programowania w assemblerze jest wystarczający i w pełni wspiera programistę.
- Dissasembly (**PL:** Dezasemblacja / **UA:** / **RU:**):
  Zakładka ta pokazuje kod podczas dezasemblacji programu, zawiera wszystkie informacje związane z działaniem aktualnego programu. Można dostrzec wszystkie instrukcje skoku miedzy procedurami/funkcjiami jak i zweryfikować kompilację programu.
- Memory (**PL:** Pamięć / **UA:** / **RU:**):
  Zakładka wyświetla wartości z pamięci operacyjnej. Można dzieki niej sprawdzić zawartość pamięci. Jest pomocna podczas obsługi stosu i tablic dynamicznych wielowymiarowych.
- Locals (**PL:** Lokalne / **UA:** / **RU:**):
  Zakładka wyświetla aktualne zmienne które w danym momencie są dostępne w danym zasięgu. Proramista nie może tam ręcznie dodawać zmiennych.

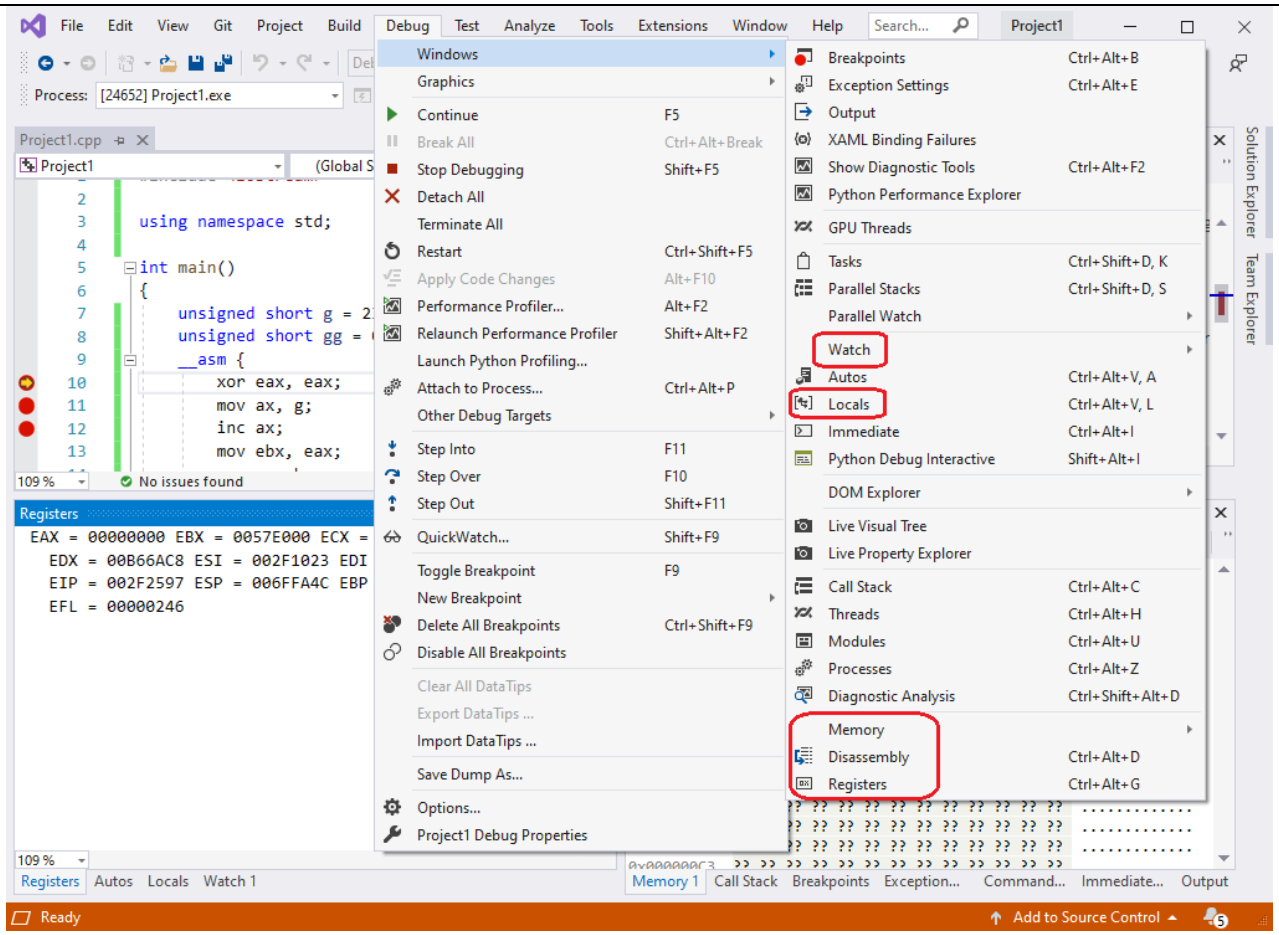
- Watch (**PL:** Wyrażenie kontrolne / **UA:** / **RU:**):
  Zakładka,która działa podobnie jak zakłądka *Locals* jednakże jest w pełni zarządzalna przez programistę. Można do niej ręcznie dodawać zmienne i nawet funkcje w celu obserwowania zmian wartości.

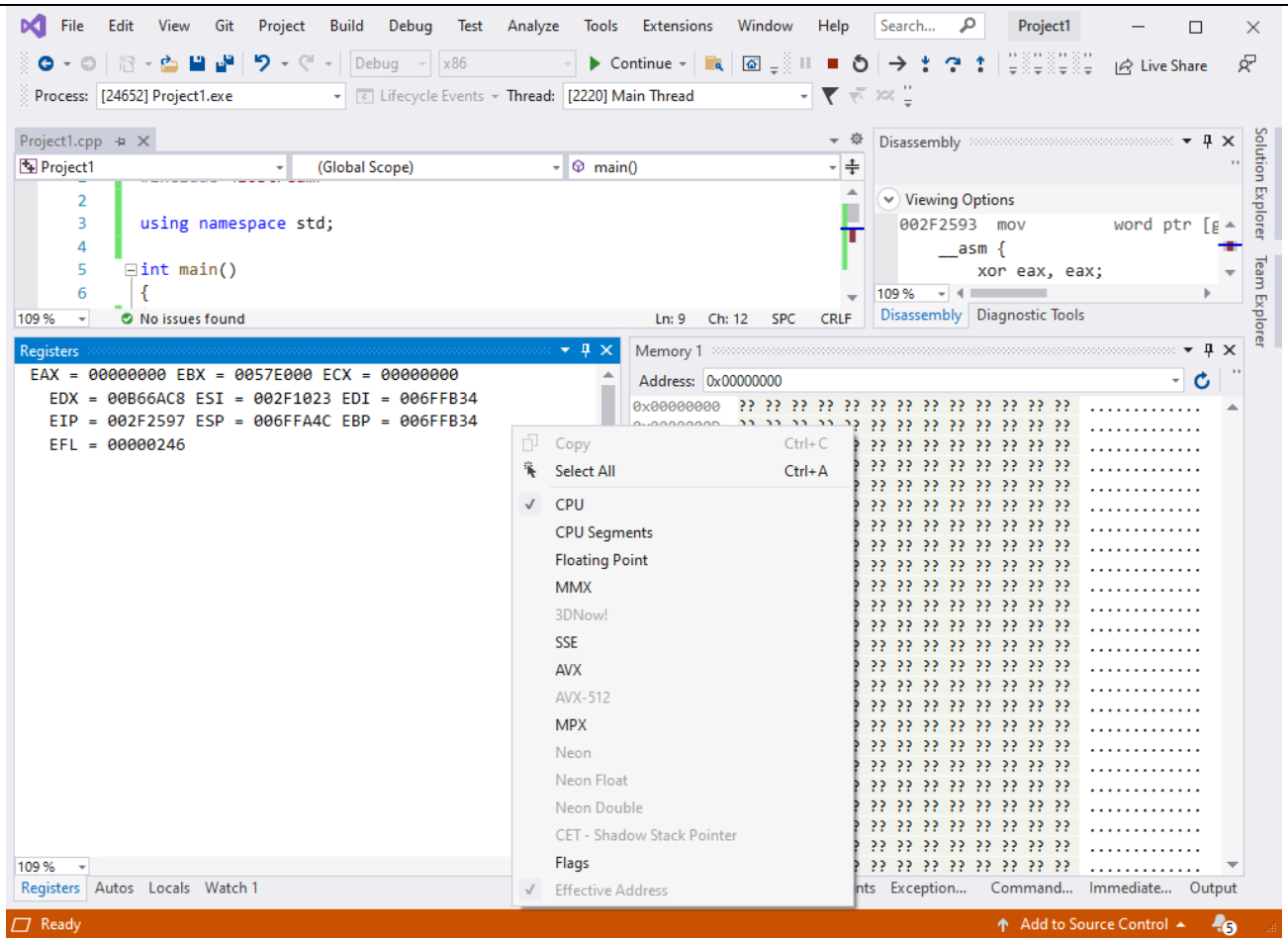
Przyciski odpowiadające za krokowe przechodzenie między instrukcjami.

Podczas pierwszego uruchomienia wyżej opisane zakładki/okna nie są domyślnie otwarte, dlatego poniżej zaprezentowany jest obrazek z lokalizajcą tych zakłądek/okien aby można było ręcznie je otworzyć. Aby je otworzyć należy podczas debugowania wejść w **Debug -> Windows** (**PL:** Debugowanie -> Okna / **UA:** / **RU:**) i wybrać te, które są w danym momencie przydatne. Oczywiście wszystkim tym zakładkom/oknom są przypisane odpowiednie skróty.

Debugowanie kodu!
By rozpocząć debugowanie należy wejść w opcję **Debug -> Start Debugging** (**PL:** Debugowanie -> Rozpocznij debugowanie / **UA:** / **RU:**)

- **Step over (F10)** (**PL:** Przekrocz nad / **UA:** / **RU:**) - pozwala na przejście do następnej linii w bieżącym kontekście, wykonuje jedną instrukcję, nieważne czy jest to przypisanie wartości do zmiennej, czy wywołanie funkcji albo procedury.
- **Step into (F11)** (**PL:** Wkrocz do / **UA:** / **RU:**) - działa podobnie jak **Step over**, z tym że można wejść również do wnętrza wywoływanych podprogramów. Pozwala na wykonanie kodu rzeczywiście linijka po linijce - z "wchodzeniem" do wnętrza wszystkich metod/funkcji/procedur.
- **Step out (Shift + F11)** (**PL:** Wyjdź / **UA:** / **RU:**) - polecenie to wykonuje cały kod w bieżącej funkcji, a następnie zatrzymuje się przy następnej instrukcji (jeśli istnieje). Innymi słowy, pozwala wyjść z bieżącej funkcji w jednym kroku.
- **Continue (F5)** (**PL:** Kontynuuj / **UA:** / **RU:**) - to polecenie wznawia program i kontynuuje jego normalne uruchamianie, bez wstrzymywania, chyba że trafi na inny punkt wstrzymania.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Windows
Graphics
Continue F5
Break All Ctrl+Alt+Break
Stop Debugging Shift+F5
Detach All
Terminate All
Restart Ctrl+Shift+F5
Apply Code Changes Alt+F10
Performance Profiler... Alt+F2
Relaunch Performance Profiler Shift+Alt+F2
Launch Python Profiling...
Attach to Process... Ctrl+Alt+P
Other Debug Targets
Step Into F11
Step Over F10
Step Out Shift+F11
QuickWatch... Shift+F9
Toggle Breakpoint F9
New Breakpoint
Delete All Breakpoints Ctrl+Shift+F9
Disable All Breakpoints
Clear All DataTips
Export DataTips ...
Import DataTips ...
Save Dump As...
Options...
Project1 Debug Properties
Breakpoints Ctrl+Alt+B
Exception Settings Ctrl+Alt+E
Output
XAML Binding Failures
Show Diagnostic Tools Ctrl+Alt+F2
Python Performance Explorer
GPU Threads
Tasks Ctrl+Shift+D, K
Parallel Stacks Ctrl+Shift+D, S
Parallel Watch
Watch
Autos Ctrl+Alt+V, A
Locals Ctrl+Alt+V, L
Immediate Ctrl+Alt+I
Python Debug Interactive Shift+Alt+I
DOM Explorer
Live Visual Tree
Live Property Explorer
Call Stack Ctrl+Alt+C
Threads Ctrl+Alt+H
Modules Ctrl+Alt+U
Processes Ctrl+Alt+Z
Diagnostic Analysis Ctrl+Shift+Alt+D
Memory
Disassembly Ctrl+Alt+D
Registers Ctrl+Alt+G
Project1.cpp
using namespace std;
int main()
{
unsigned short g = 2
unsigned short gg =
__asm {
xor eax, eax;
mov ax, g;
inc ax;
mov ebx, eax;
No issues found
Registers
EAX = 00000000 EBX = 0057E000 ECX =
EDX = 00B66AC8 ESI = 002F1023 EDI
EIP = 002F2597 ESP = 006FFA4C EBP
EFL = 00000246
Registers Autos Locals Watch 1
Memory 1 Call Stack Breakpoints Exception... Command... Immediate... Output
Ready
Add to Source Control



Brakuje już tylko wyświetlenia dodatkowych rejestrów. Aby to zrobić podczas trybu debugowania należy w zakładce Registers na pustym białym tle kliknąć prawym przyciskiem myszki i zaznaczyć potrzebne rejestry.

Na początek do działania na liczbach stałoprzecinkowych na procesorze wystarczy:

- CPU
- CPU Segments (**PL:** Segmenty CPU / **UA:** / **RU:**):
- Flags (**PL:** Flagi / **UA:** / **RU:**):

Widoki pozostałych rejstrów również można włączyć jednakże nie będą one potrzebne na tym etapie prowadzenia zajęć. Dopiero podczas drugiej części zajęć czyli operacji na liczbach zmiennoprzecinkowych wymagane będzie włączenie Floating Point i później rozszerzeń wektorowych MMX/SSE/AVX.

Zadania do wykonania.

1. Proszę przepisać działający kod z wcześniejszego zrzutu ekranu i przejść krok po kroku patrząc i analizując zmiany stanów poszczególnych rejestrów.

2. Proszę przepisać kod z następującego zrzutu ekranu i zgłosić ten fakt prowadzącemu w celu wspólnej analizy w trakcie zajęć:

```
4
5   int main()
6   {
7       unsigned int a = 10; //??
8       unsigned int b = 20; //??
9       unsigned short g = 10;
10      unsigned short gg;
11      __asm
12      {
13          push ebx;
14          xor ecx, ecx;
15          mov ebx, ecx;
16          mov eax, 0xaaff; //?? zapisc C++
17          mov eax, 0aaffh; //?? zapis asm
18          mov ebx, eax;
19          inc eax; //??
20          inc eax;
21          dec ecx;
22          dec eax;
23          cmp ebx, 0;
24          jnz skok; // jz skok
25          mov ax, g;
26          inc ax;
27          mov ebx, eax;
28          mov gg, bx;
29      skok:   mov eax, a; //??
30          mov ebx, b; //??
31          add eax, b; //??
32          cmp eax, ebx;
33          sub eax, b;
34          mul ecx;
35          div eax; //??
36          pop ebx;
37      }
38      cout << a << "; " << b << endl;
39      system("PAUSE");
40      return 0;
41  }
42
```

UWAGA: Niektóre instrukcje są celowo wprowadzone z błędem, należy je wtedy ustawić jako komentarz lub spróbować poprawić na prawidłowe instrukcje razem z prowadzącym zajęcia. W przypadku błędnych instrukcji należy zwrócić uwagę na komunikaty jakie przekazuje kompilator.

Zadanie domowe: Proszę na następne zajęcia powtórzyć wiedzę zdobytą na wykładzie i na laboratorium.