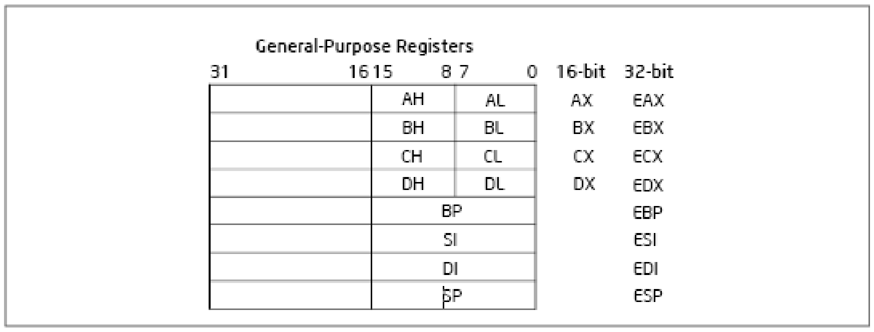
Computer Organization and Assembly Language

Quick reference guide for Instructors

32-bit Inline Assembly Programming in Visual Studio 2019

Introduction to 32-bit programming

Intel 8386 was the first 32-bit processor of the Intel family of microprocessors. It has 32-bit address and data buses. The protected mode was also introduced in this processor that limits the operation of user programs. It doesn’t allow user programs to go beyond their memory limits assign to them and directly access IO devices. The size of existing registers, except the segment registers, were extended to 32-bit as shown below.



Addressing Modes

The addressing modes are the same.

In the 80386 and above, any register from EAX, EBX, ECX, EDX, EBP, EDI, ESI may be used in register indirect addressing mode. (Example: The MOV [EDX], CL instruction copies the byte sized contents of register CL into the data segment memory location addressed by EDX)

In the 80386 and above, any two registers from EAX, EBX, ECX, EDX, EBP, EDI, or ESI may be combined to generate the memory address. (Example: The MOV [EAX+EBX], CL instruction copies the byte sized contents of register CL into the data segment memory location addressed by EAX plus EBX.)

The 80386 and above use any 32-bit register except ESP to address memory. (Example: MOV AX, [ECX] or MOV AX, ARRAY[EBX]. The first instruction loads AX from the data segment address formed by ECX plus 4. The second instruction loads AX from the data segment memory location ARRAY plus the contents of EBX.

Inline Assembler *(Microsoft Specific)*

Assembly language serves many purposes, such as improving program speed, reducing memory needs, and controlling hardware when writing kernel modules. You can use the inline assembler to embed assembly-language instructions directly in your C and C++ source programs without extra assembly and link steps. The inline assembler is built into the compiler, so you don't need a separate assembler such as the Microsoft Macro Assembler (MASM). Because the inline assembler doesn't require separate assembly and link steps, it is more convenient than a separate assembler. Inline assembly code can use any C or C++ variable or function name that is in scope, so it is easy to integrate it with your program's C and C++ code.

And because the assembly code can be mixed with C and C++ statements, it can do tasks that are cumbersome in C or C++ alone.

The__asm keyword invokes the inline assembler and can appear wherever a C or C++ statement is legal. It cannot appear by itself. It must be followed by an assembly instruction, a group of instructions enclosed in braces, or, at the very least, an empty pair of braces.

Example

```
__asm mov eax,ebx
```

or

```
__asm

{

Mov eax,ebx


}
```

Using and Preserving Registers in Inline Assembly *(Microsoft Specific)*

In general, you should not assume that a register will have a given value when an __asm block begins. Register values are not guaranteed to be preserved across separate __asm blocks. If you end a block of inline code and begin another, you cannot rely on the registers in the second block to retain their values from the first block. An __asm block inherits whatever register values result from the normal flow of control.

When using __asm to write assembly language in C/C++ functions, you don't need to preserve the EAX, EBX, ECX, EDX,ESI, or EDI registers. You should preserve other registers you use (such as DS,SS,ESP, EBP,and flags registers) for the scope of the __asm block.

Example#1: Program to add two integer numbers using the inline assembly.

```
#include<iostream>
using namespace std;


int main(void)
{
	int a = 10, b = 20, c = 0;
	__asm
		{
			mov ebx,a
			add ebx,b
			mov c,ebx
		}
		cout << "Sum of " << a << " and " << b << " is " << c << endl;

}
```

Example#2: Program to access array using inline 32-bit assembly. Both codes are incrementing every element of the array.

```
#include<iostream>
using namespace std;

int array1[] = {1,2,3,4,5};

int main(void)
{
	__asm
	{
		push ebp
		mov ebp, offset array1
		mov esi, 0
		mov ecx, 5
	l1:
		inc dword ptr[ebp + esi]
		add esi, 4
		loop l1
		pop ebp
	}
	for (int i = 0; i < 5; i++)
		cout << array1[i];
}
```

```
#include<iostream>
using namespace std;

int array1[] = {1,2,3,4,5};

int main(void)
{

	for (int i = 0; i < 20; i=i+4)
	{

		__asm
		{
		mov esi,i
		inc dword ptr [array1+esi]

		}
	}
	for (int i = 0; i < 5; i++)
		cout << array1[i];
}
```

Example#3: Program to rotate right each element of the array using 1) C++ and 2) inline assembly.

Rotate left in C++

```
#include<iostream>
using namespace std;
int main()
{
	int array1[] = { 1,2,3,4,5 };
	int DROPPED_MSB;
for (int i = 0; i < 5; i++)
{
DROPPED_MSB = (array1[i] << 1);
array1[i] = (array1[i] << 1) | DROPPED_MSB;
}
for (int i = 0; i < 5; i++)
{
cout << array1[i] << endl;
}
return 0;
}
```

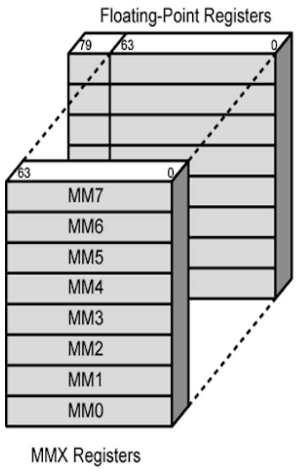
Rotate left using inline assembly

```
#include<iostream>
using namespace std;
int main(void)
{
	int array1[] = { 1,2,3,4,5 };
	for (int i = 0; i < 20; i=i+4)
	{
	__asm
	{
	mov eax,i
	rol dword ptr [array1+eax],1
	}
	}
	for (int i = 0; i < 5; i++)
	{
		cout << array1[i] << endl;
	}
}
```

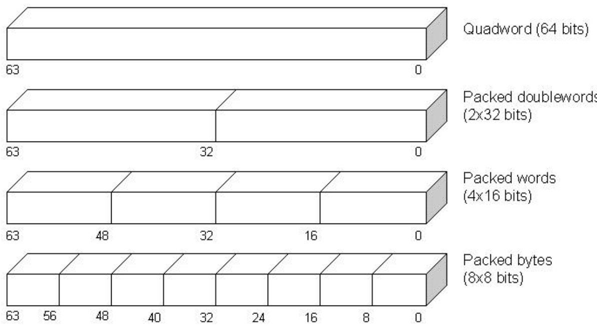
Multimedia Extension (MMX)

MMX refers to a feature that exist in the Intel processors, which can boost up multimedia software, by executing integer instructions on several data parallelly. It allows single instruction to work on multiple data (SIMD). Though the processor supports the acceleration, it does not mean your application can make use of it automatically. Hence, your application must be manually written in order to take advantage of it.

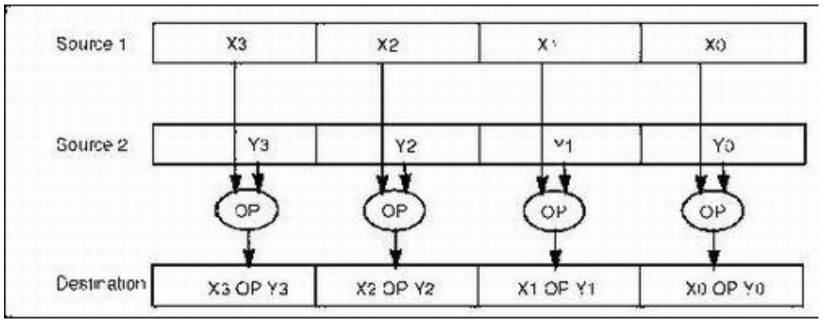
So far, we have seen general purpose registers. Besides that, we also have another 8 80-bit registers located in the FPU (Floating Point Unit) named as mm0, mm1, mm2, mm3, mm4, mm5, mm6 and mm7 as shown below. We can make use of these 8 registers in FPU if we have MMX feature in our processor. Only least 64 bits of these registers are accessible for mmx operations.



The MMX registers can store 8 bytes, 4 words, 2 double words and 1 quad word as shown in the image below.



MMX instructions performs operations on the corresponding bytes, words or doublewords store in registers as shown below.



MMX™ Instruction Set Summary

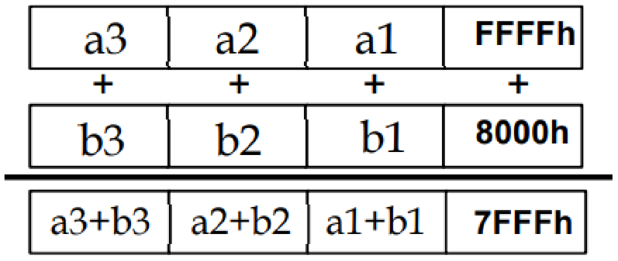
The instructions and corresponding mnemonics in the table below are grouped by function categories. If an instruction supports multiple data types—byte (B), word (W), doubleword (DW), or quadword (QW), the datatypes are listed in brackets. Only one data type may be chosen for a given instruction. For example, the base mnemonic PADD (packed add) has the following variations: PADDB, PADDW, and PADDD. The number of opcodes associated with each base mnemonic is listed.

Category	Mnemonic	Number of Different Opcodes	Description
Arithmetic	PADD[B,W,D]	3	Add with wrap-around on [byte, word, doubleword]
	PADDS[B,W]	2	Add signed with saturation on [byte, word]
	PADDUS[B,W]	2	Add unsigned with saturation on [byte, word]
	PSUB[B,W,D]	3	Subtraction with wrap-around on [byte, word, doubleword]
	PSUBS[B,W]	2	Subtract signed with saturation on [byte, word]
	PSUBUS[B,W]	2	Subtract unsigned with saturation on [byte, word]
	PMULHW	1	Packed multiply high on words
	PMULLW	1	Packed multiply low on words
	PMADDWD	1	Packed multiply on words and add resulting pairs
Comparison	PCMPEQ[B,W,D]	3	Packed compare for equality [byte, word, doubleword]
	PCMPGT[B,W,D]	3	Packed compare greater than [byte, word, doubleword]
Conversion	PACKUSWB	1	Pack words into bytes (unsigned with saturation)
	PACKSS[WB,DW]	2	Pack [words into bytes, doublewords into words] (signed with saturation)
	PUNPCKH [BW,WD,DQ]	3	Unpack (interleave) high-order [bytes, words, doublewords] from MMX™ register
	PUNPCKL [BW,WD,DQ]	3	Unpack (interleave) low-order [bytes, words, doublewords] from MMX register
Logical	PAND	1	Bitwise AND
	PANDN	1	Bitwise AND NOT
	POR	1	Bitwise OR
	PXOR	1	Bitwise XOR

Shift	PSLL[W,D,Q]	6	Packed shift left logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRL[W,D,Q]	6	Packed shift right logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRA[W,D]	4	Packed shift right arithmetic [word, doubleword] by amount specified in MMX register or by immediate value
Data Transfer	MOV[D,Q]	4	Move [doubleword, quadword] to MMX register or from MMX register
FP & MMX State Mgmt	EMMS	1	Empty MMX state

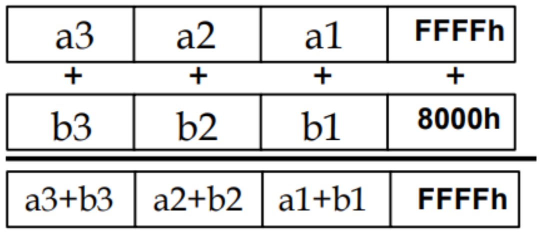
Instruction Examples

The following section will briefly describe five examples of MMX instructions. For illustration, the data type shown in this section will be the 16-bit word data type; most of these operations also exist for 8-bit or 32-bit packed data types. The following example shows a packed add word with wrap around. It performs four additions of the eight, 16-bit elements, with each addition independent of the others and in parallel. In this case, the right-most result exceeds the maximum value representable in 16-bits—thus it wraps around. This is the way regular IA arithmetic behaves. FFFFh + 8000h would be a 17-bit result. The 17th bit is lost because of wrap around, so the result is 7FFFh.



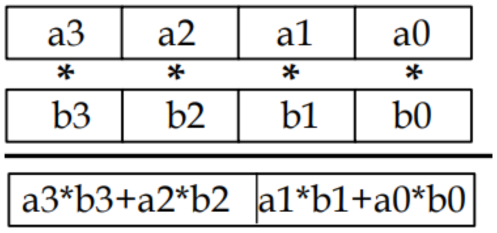
PADD[W]: Wrap-around Add

The following example is for a packed add word with unsigned saturation. This example uses the same data values from before. The right-most add generates a result that does not fit into 16 bits; consequently, in this case saturation occurs. Saturation means that if addition results in overflow or subtraction results in underflow, the result is clamped to the largest or the smallest value representable. For an unsigned, 16-bit word, the largest and the smallest representable values are FFFFh and 0x0000; for a signed word the largest and the smallest representable values are 7FFFh and 0x8000.



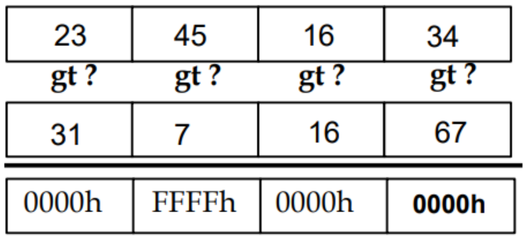
PADDUS[W]: Saturating Arithmetic

The PMADD instruction starts from a 16-bit, packed data type and generates a 32-bit packed, data type result. It multiplies all the corresponding elements generating four 32-bit results and adds the two products on the left together for one result and the two products on the right together for the other result. To complete a multiply-accumulate operation, the results would then be added to another register which is used as the accumulator.



PMADDWD: 16b x 16b -> 32b Multiply Add

The following example is a packed parallel compare. This example compares four pairs of 16-bit words. It creates a result of true (FFFFh), or false (0000h). This result is a packed mask of ones for each true condition, or zeros for each false condition. The following example shows an example of a compare “greater than” on packed word data. There are no new condition code flags, nor are any existing IA condition code flags affected by this instruction.



PCMPGT[W]: Parallel Compares

Example#4: Program to add two byte arrays using MMX instructions.

```
#include<iostream>
using namespace std;
int main()
{
	char array1[8] = { 1,2,3,4,5,6,7,8 };
	char array2[8] = { 6,7,8,9,10,11,12,13 };
	char array3[8];

	__asm
	{
		movq mm0,[array1]
		movq mm1,[array2]
		paddb mm0,mm1
		movq  [array3],mm0
	}
	for (int i = 0; i < 8; i++)
	{
		cout <<"Sum of "<<(uint8)array1[i] <<" and "<< (int)array2[i]<<" is "<<
(int)array3[i]<< endl;
	}
	return 0;
}
```

Example#5: Program to shift left all elements of int array using MMX instructions.

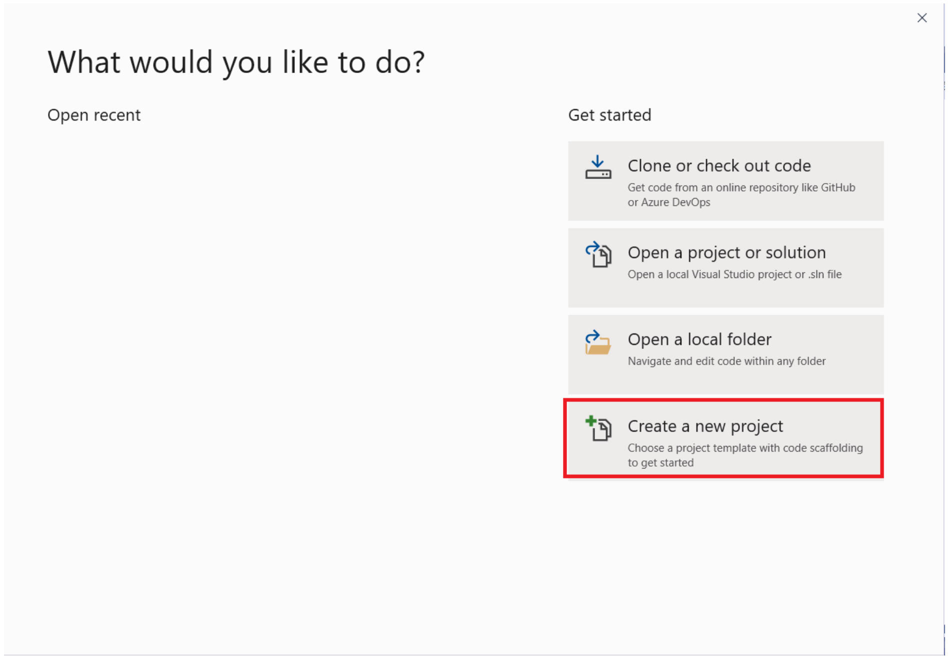
```
#include<iostream>
using namespace std;
int main()
{
         int array1[6] = { 1,2,3,4,5,6 };
         int array2[6];
        for(int i=0;i<24;i=i+8)
        __asm
        {
                mov esi,i
                movq mm0,[array1+esi]
                pslld mm0,1
                movq  [array2+esi],mm0
        }
        for (int i = 0; i < 6; i++)
        {
                cout << "Sum of " << (int)array2[i] << endl;
        }
        return 0;
}
```

Computer Organization and Assembly Language

Step#1:

Open Visual Studio 2019

Step#2: Click on "Create a new project"



What would you like to do?

Open recent

Get started

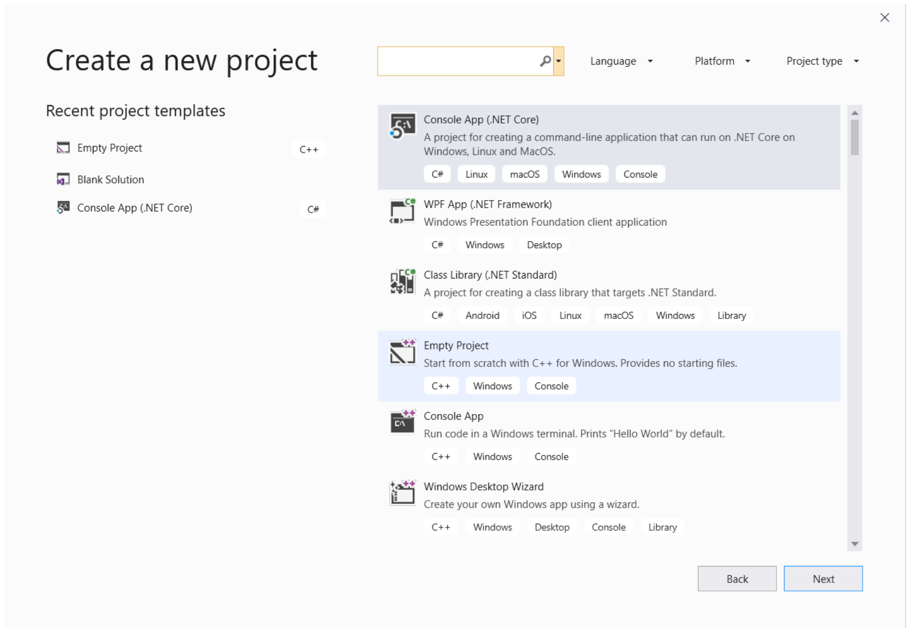
Clone or check out code
Get code from an online repository like GitHub or Azure DevOps

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

Step#3: Click on C++ under "Empty Project" and press Next



Create a new project

Recent project templates

Empty Project	C++
Blank Solution	
Console App (.NET Core)	C#

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

WPF App (.NET Framework)
Windows Presentation Foundation client application
C# Windows Desktop

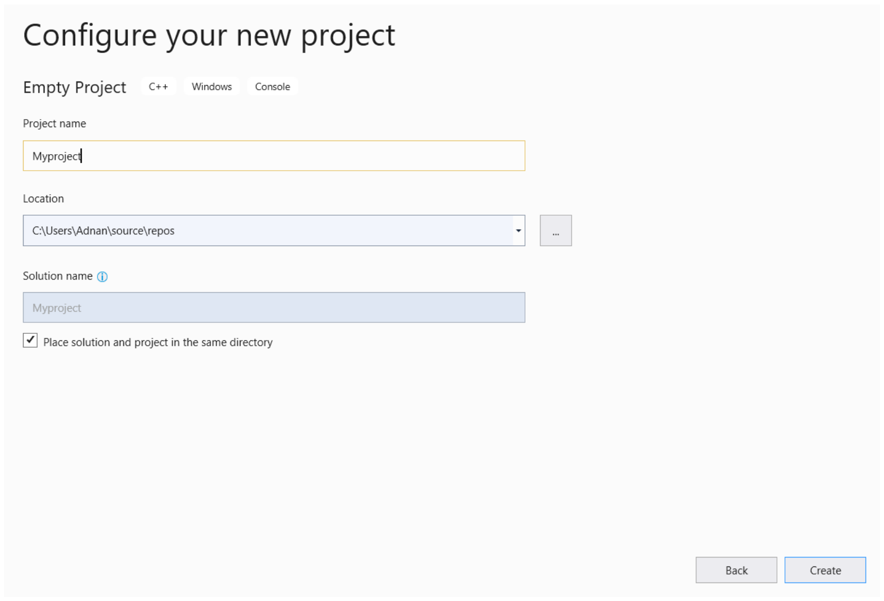
Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

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

Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

Step#4: Write your project name (e.g., Myproject) and click on Create



Configure your new project

Empty Project C++ Windows Console

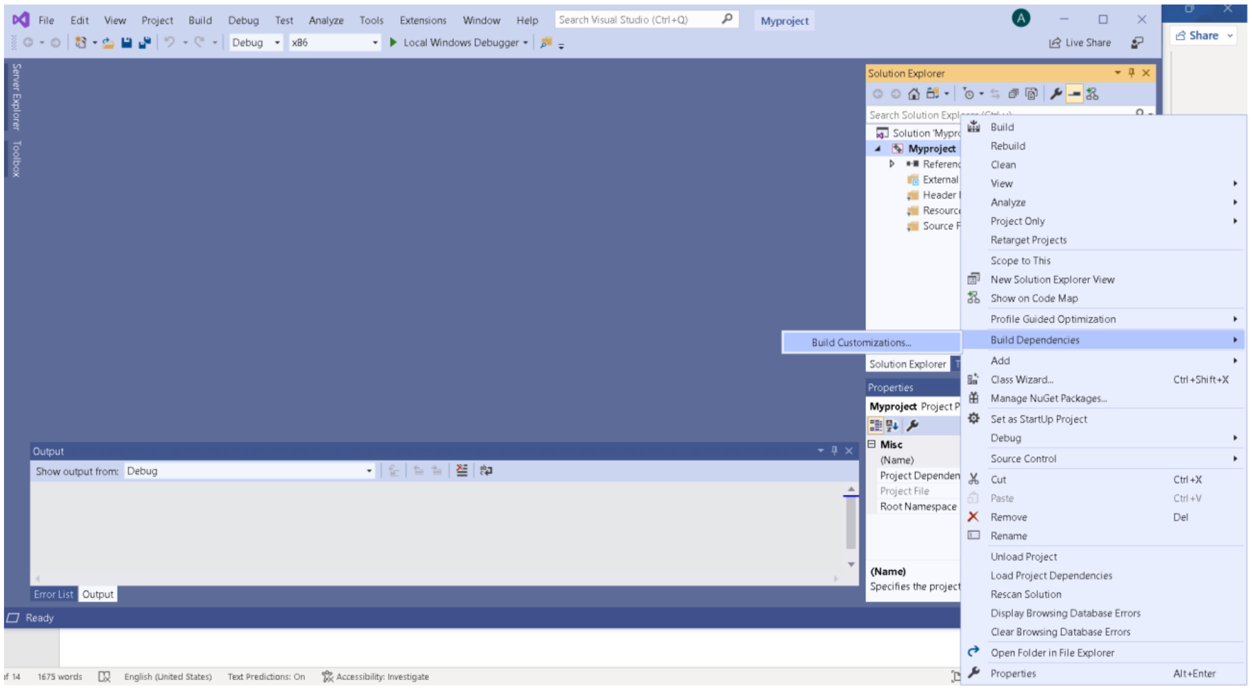
Project name: Myproject

Location: C:\Users\Adnan\source\repos

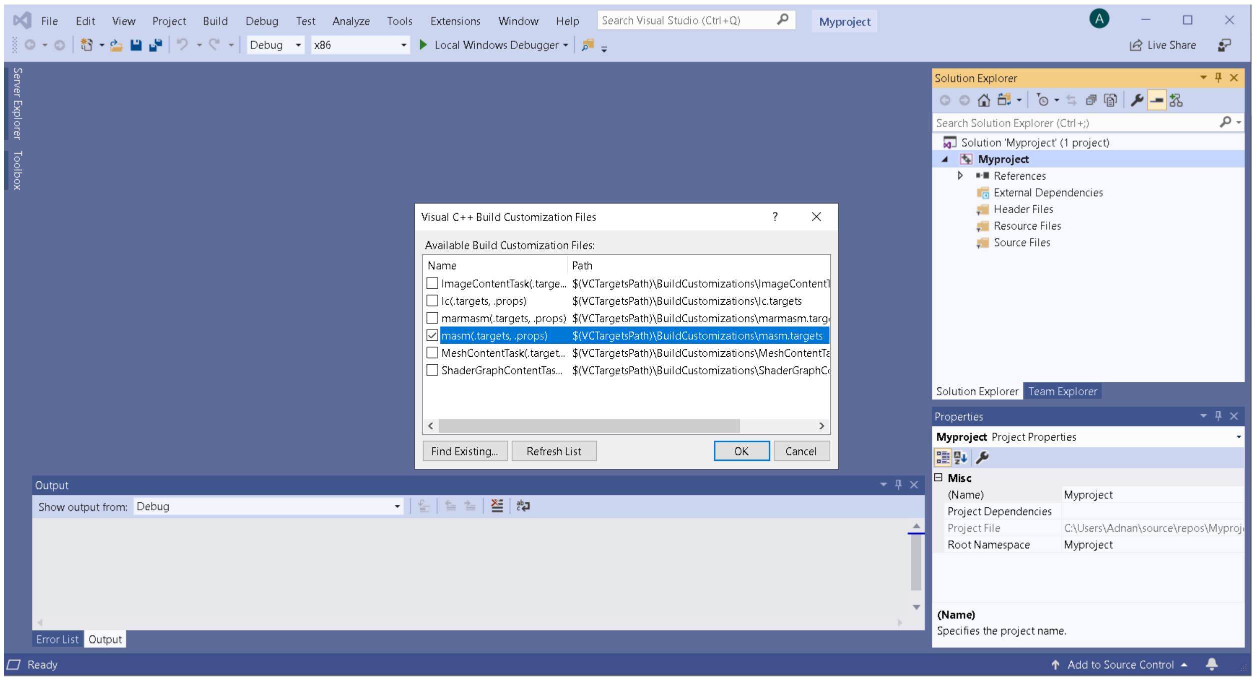
Solution name: Myproject

[✓] Place solution and project in the same directory

Step#5: Right click on project name (i.e., Myproject) → Build Dependencies →Build Customizations



Step#5: Select "masm" and press OK



Visual C++ Build Customization Files

Available Build Customization Files:

	Name	Path
☐	ImageContentTask(.targe...	$(VCTargetsPath)\BuildCustomizations\ImageContentT
☐	lc(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\lc.targets
☐	marmasm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\marmasm.targ
☑	masm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\masm.targets
☐	MeshContentTask(.target...	$(VCTargetsPath)\BuildCustomizations\MeshContentTa
☐	ShaderGraphContentTas...	$(VCTargetsPath)\BuildCustomizations\ShaderGraphC

Properties

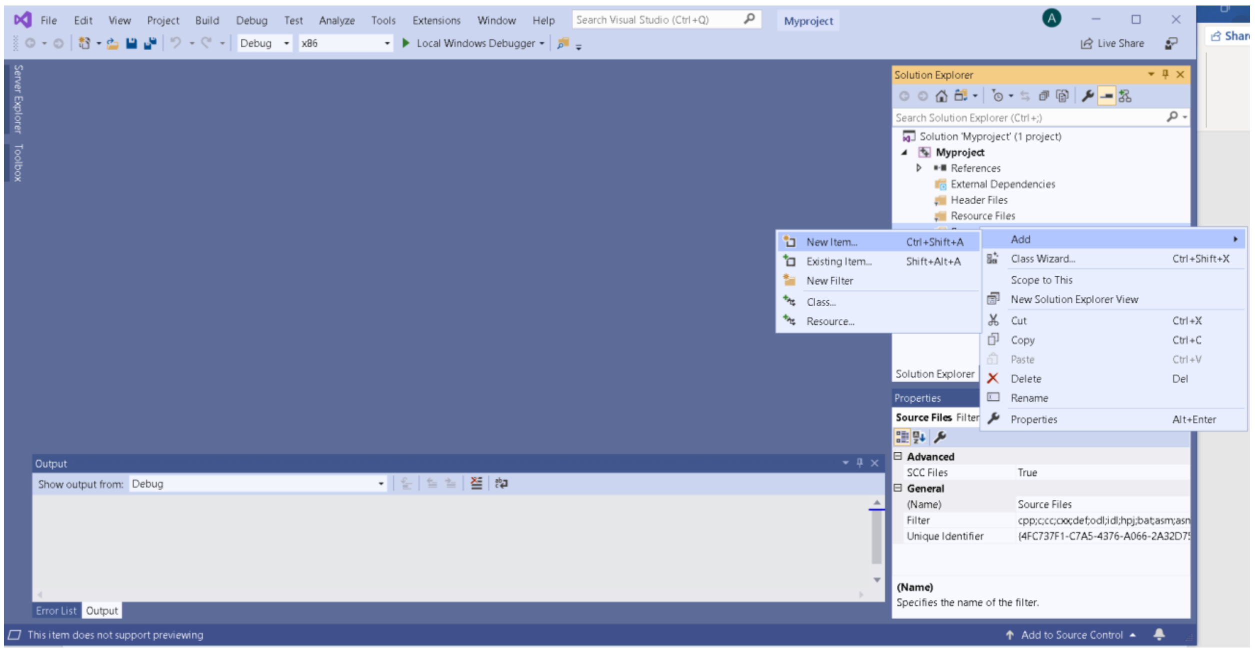
Myproject Project Properties

Misc

(Name)	Myproject
Project Dependencies	
Project File	C:\Users\Adnan\source\repos\Myproj
Root Namespace	Myproject

(Name)
Specifies the project name.

Step#6: Right click on "Source Files" → Add → New Item



Properties

Source Files Filter

Advanced

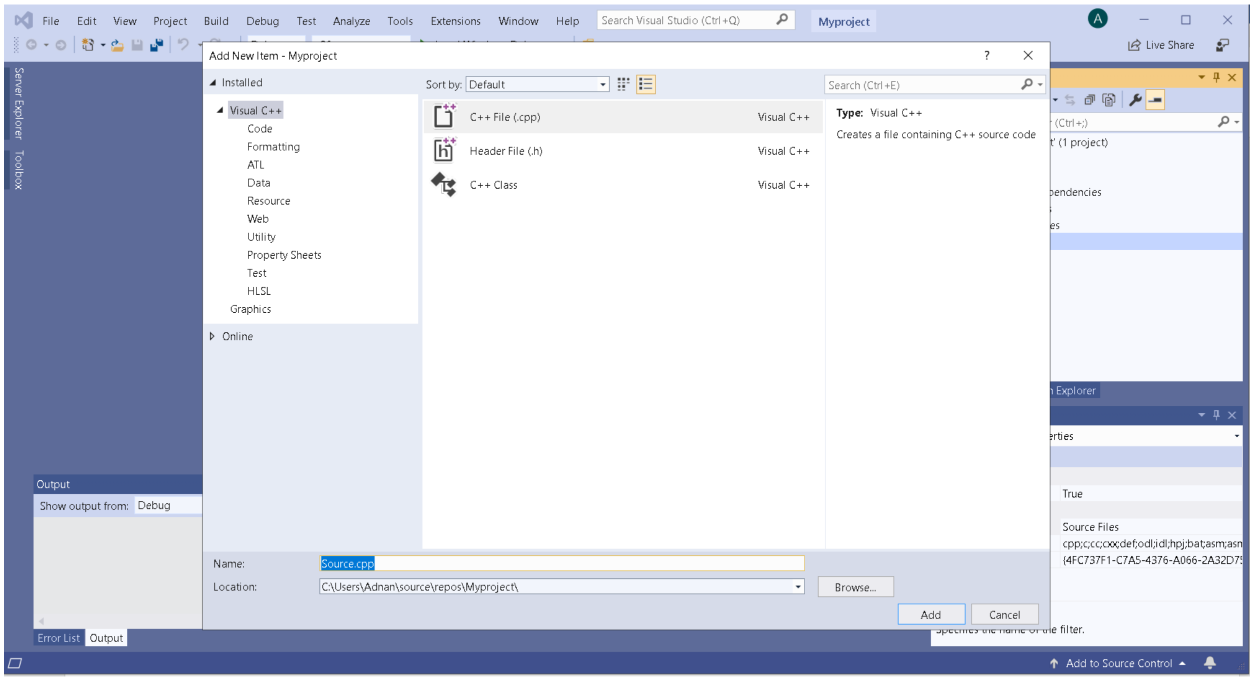
SCC Files	True

General

(Name)	Source Files
Filter	cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asn
Unique Identifier	{4FC737F1-C7A5-4376-A066-2A32D75

(Name)
Specifies the name of the filter.

Step#7: Click in "C++ File (.cpp) , write a file name and click on Add.



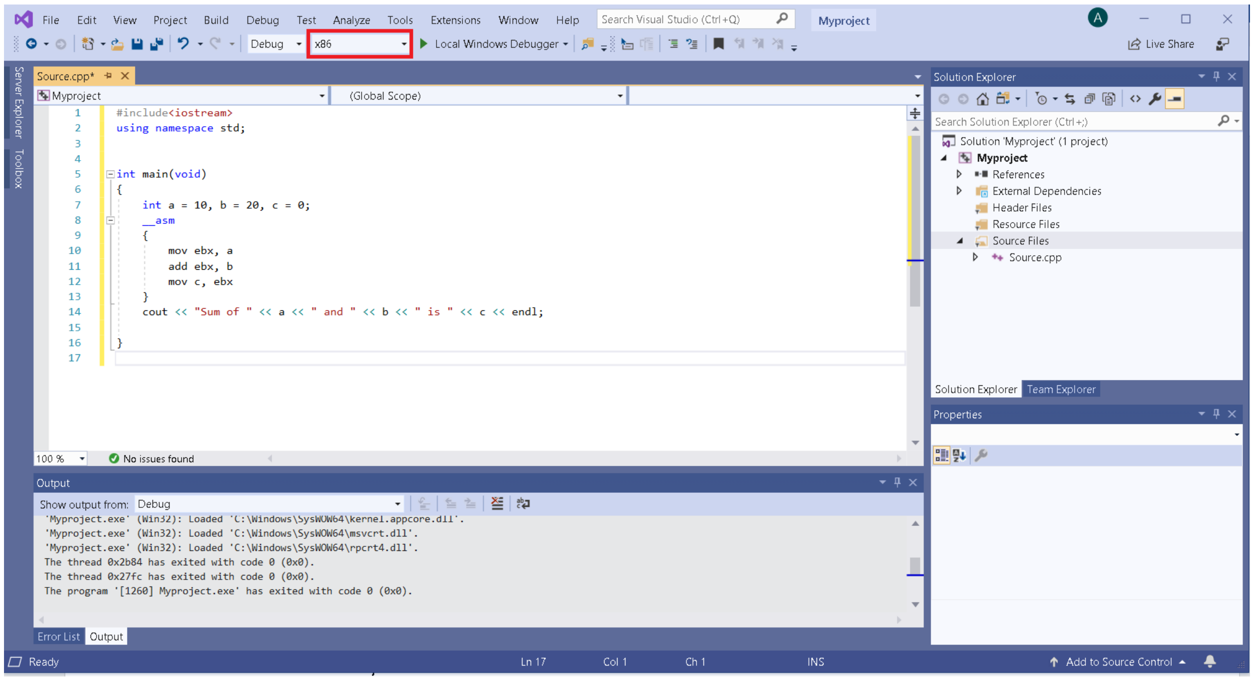
Type: Visual C++

Creates a file containing C++ source code

Name: Source.cpp

Location: C:\Users\Adnan\source\repos\Myproject\

Step#8: Paste code in Example#1 in the text window. Make sure that the environment is set to x86.

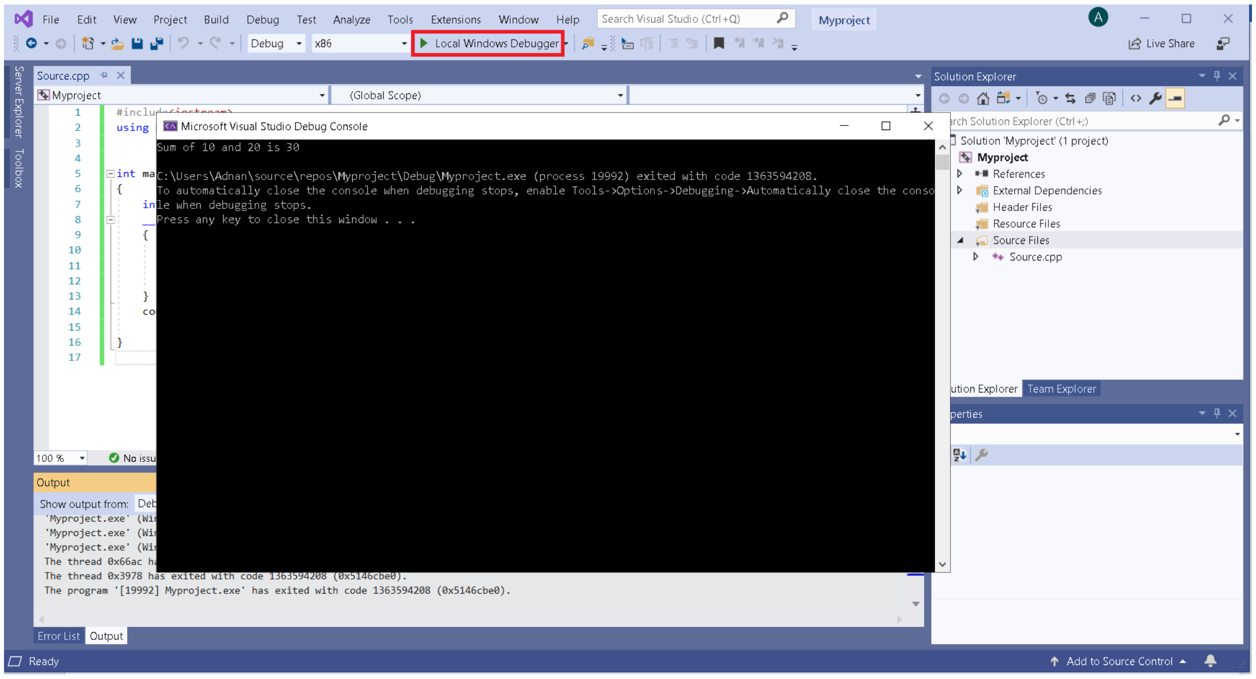


```
#include<iostream>
using namespace std;


int main(void)
{
    int a = 10, b = 20, c = 0;
    __asm
    {
        mov ebx, a
        add ebx, b
        mov c, ebx
    }
    cout << "Sum of " << a << " and " << b << " is " << c << endl;

}
```

Show output from: Debug

```
'Myproject.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel.appcore.dll'.
'Myproject.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcrt.dll'.
'Myproject.exe' (Win32): Loaded 'C:\Windows\SysWOW64\rpcrt4.dll'.
The thread 0x2b84 has exited with code 0 (0x0).
The thread 0x27fc has exited with code 0 (0x0).
The program '[1260] Myproject.exe' has exited with code 0 (0x0).
```

Step#8: Click on “Local Windows Debugger” to run the code.



```
Sum of 10 and 20 is 30

C:\Users\Adnan\source\repos\Myproject\Debug\Myproject.exe (process 19992) exited with code 1363594208.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
The thread 0x3978 has exited with code 1363594208 (0x5146cbe0).
The program '[19992] Myproject.exe' has exited with code 1363594208 (0x5146cbe0).
```