

Programming in Assembler

Laboratory manual

Exercise 8 (v.2)

MASM x64 –console application in Visual Studio 2012/2013



Exercise goal:

Students get familiarized with x64 programming in pure assembler code, using Visual Studio 2012/2013 - by building sample Windows console application.

1. x64 Assembler features.

Visual Studio 2013 contains latest Macro Assembler compiler in both x86 (32 bit) and x64 (64 bit) versions. The compiler files are ml.exe and ml64.exe respectively.

Regarding the previous releases of the assembler it is necessary to know that while writing applications for x64, a different approach is required:

- there is no inline ASM support,
- the call and return is different schema than x86 functions (as in C++), see lab 9 instruction for full details, sample is provided below:
 - up to four arguments are passed through registers RCX, RDX, R8, R9 (for Integer), XMM0-XMM3 when floating point, the above is passed on stack, in case of mixed call, arguments are passed positional-mixed, see example:

```
func3(int a, double b, int c, float d);  
// a in RCX, b in XMM1, c in R8, d in XMM3
```

- return values are returned regarding its size: 64bit in RAX, 128bit in XMM0, bigger by pointer (structure) in RAX



2. Installing & using MASM:

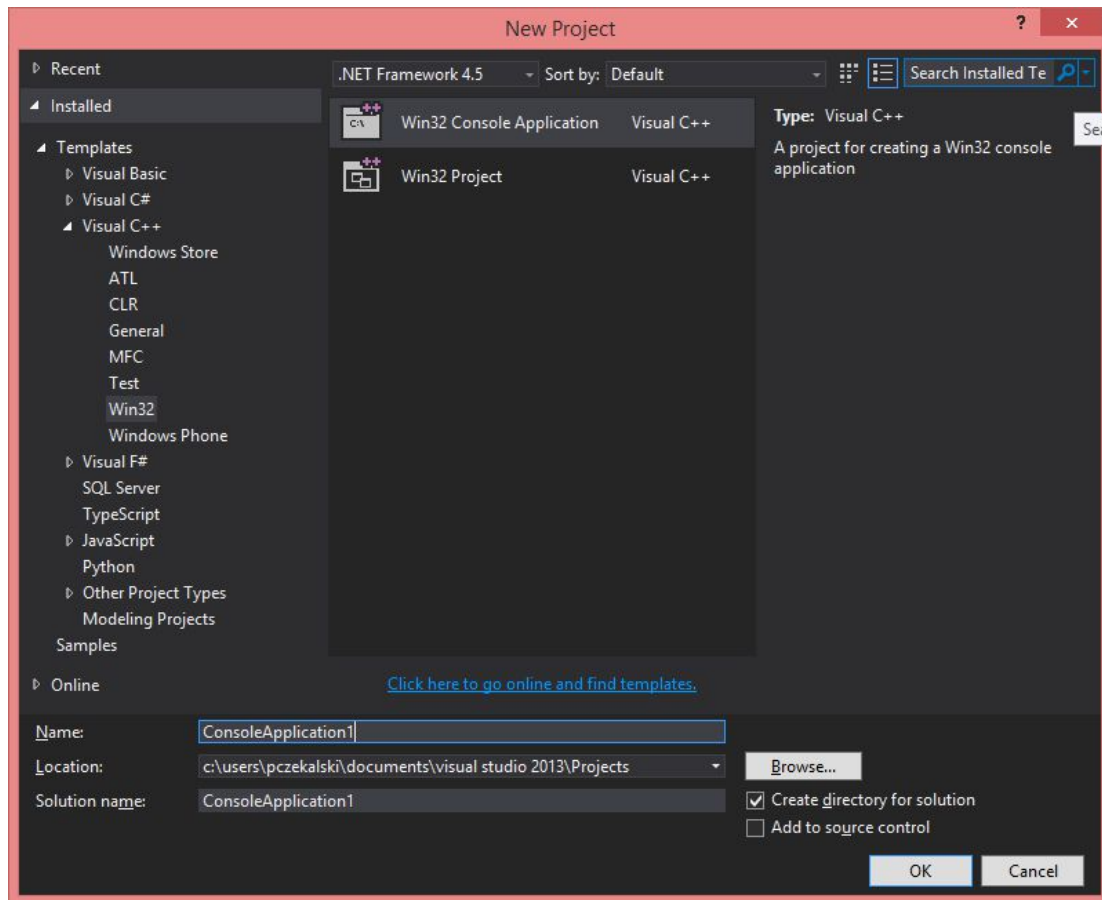
Installing the MASM does not require any additional steps than installing of the Visual Studio 2012/2013 with C++ compiler support. The useful tool to colorize the syntax of the assembler source code is available as the plugin, currently for Visual Studio 2012 only: <http://asmhighlighter.codeplex.com/>.

No additional tools are required.

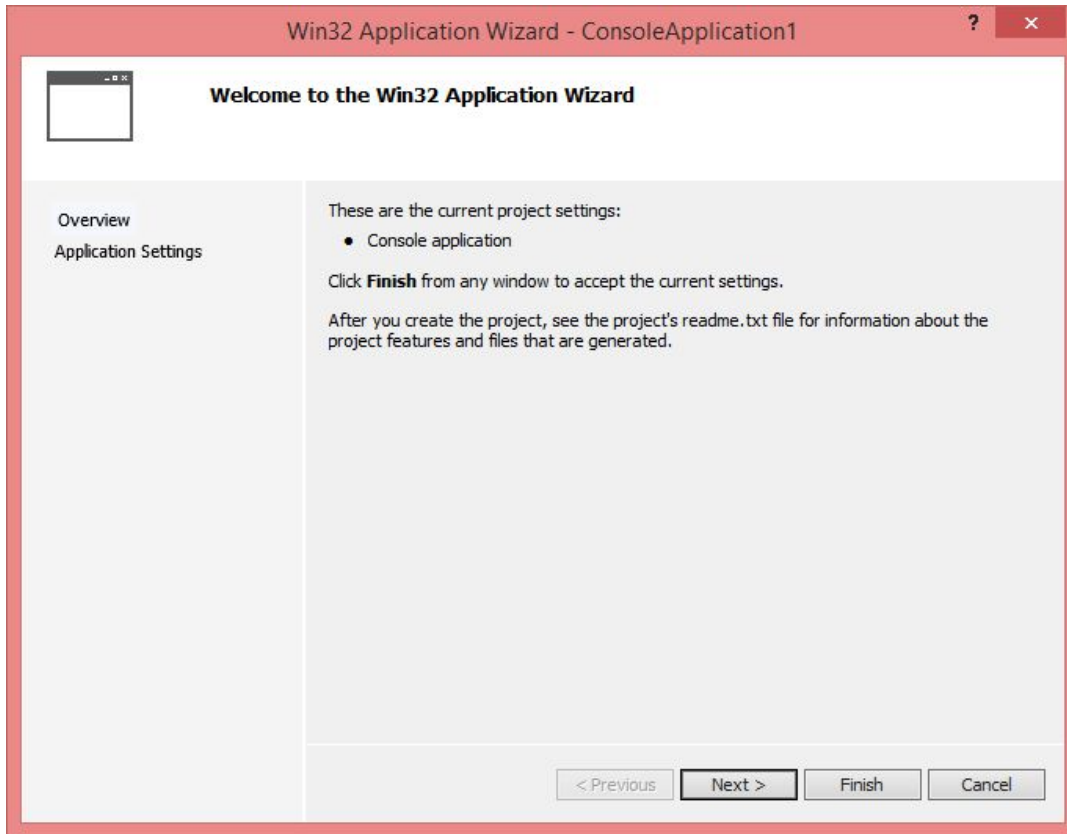
3. Creating simple application from scratch:

To start programming it is necessary to follow few basic steps.

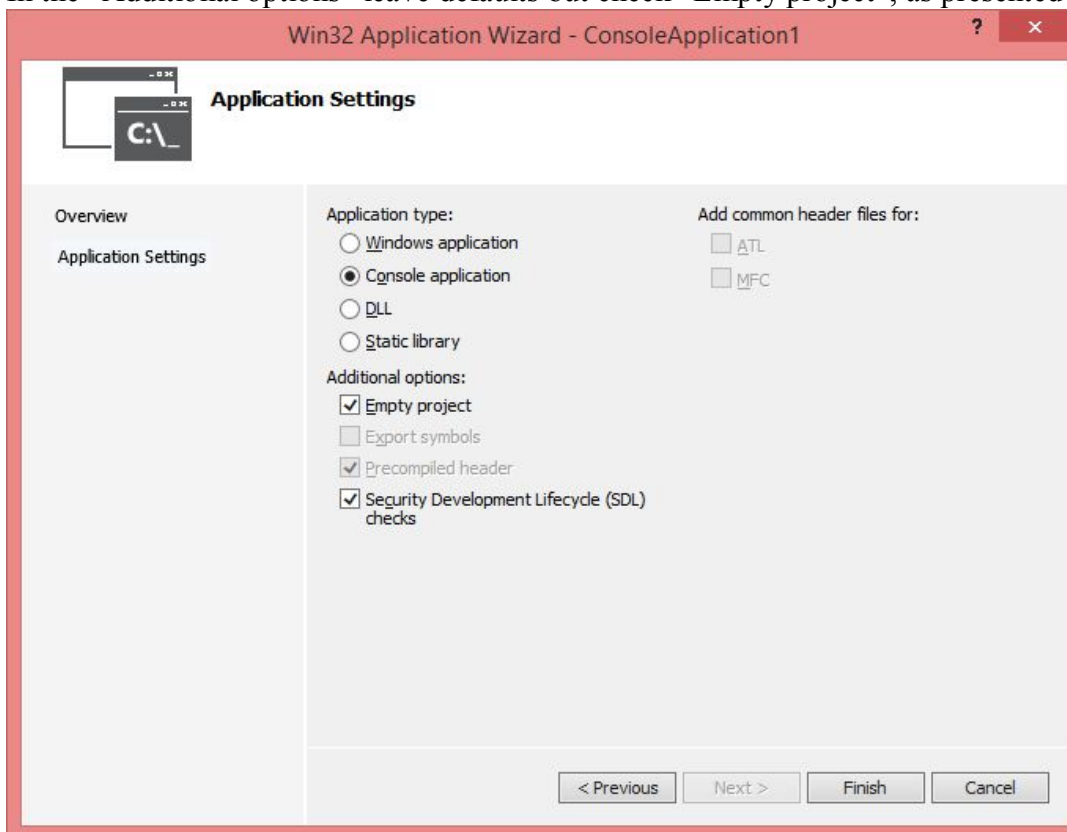
- a) Run Visual Studio and create the Win32 console application:



Click OK then in the following Wizard (see below) do not click “Finish”, click “Next” instead:



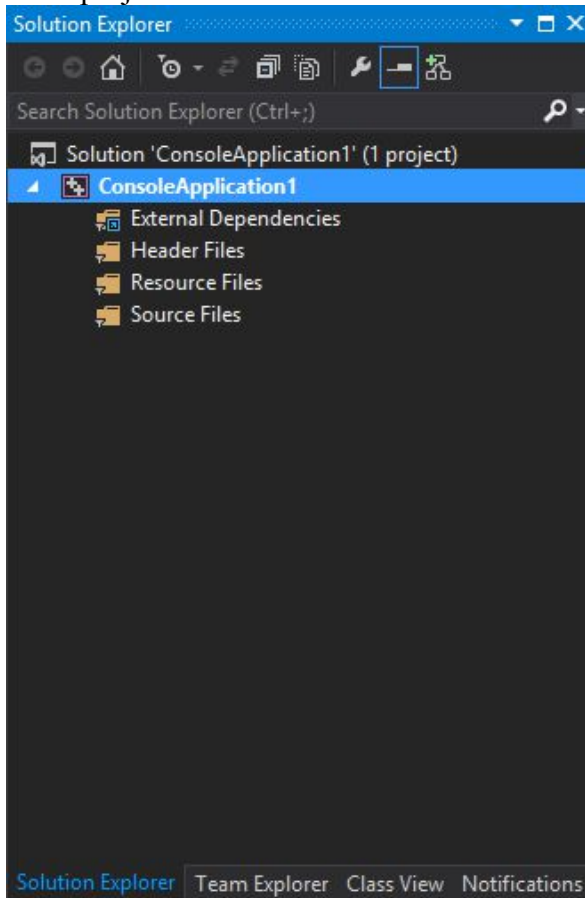
- b) Follow to the next wizard window.
 In the “Additional options” leave defaults but check “Empty project”, as presented below:



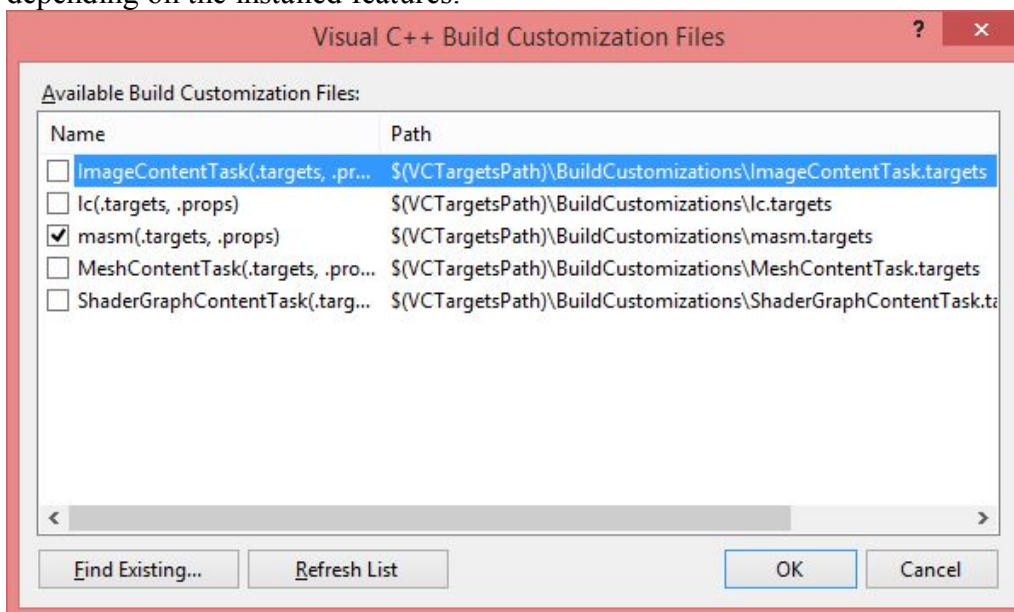
Then click “Finish” to create an empty project



c) Your project structure should look like below (see “Solution Explorer”):

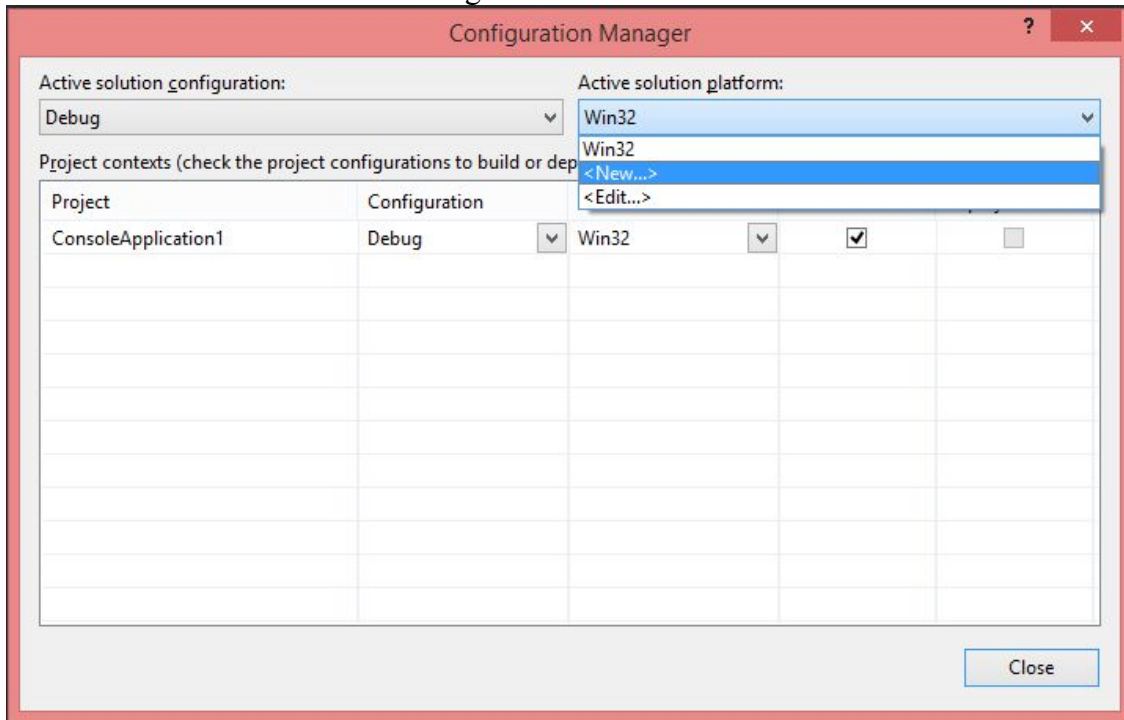


d) Customize the Build rules to enable MASM support. Click [Project>>Build Customization] in the VS menu then enable MASM support by checking the “masm(.target, .props)” – the list may vary depending on the installed features:

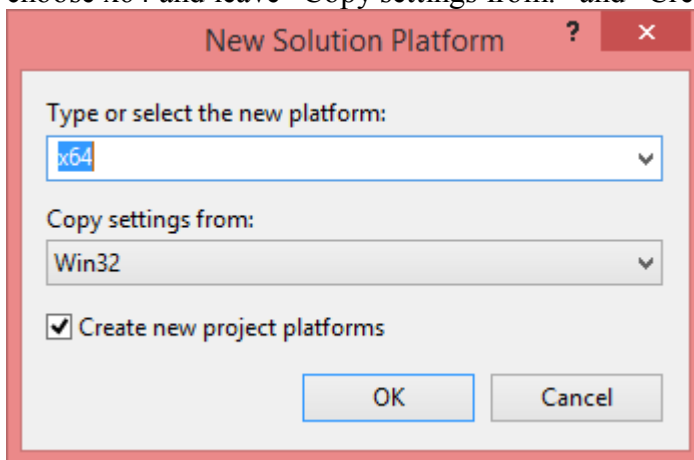




- e) Configure the project for x64. Run configuration manager (click [BUILD>>Configuration Manager] in the VS menu then add New configuration:

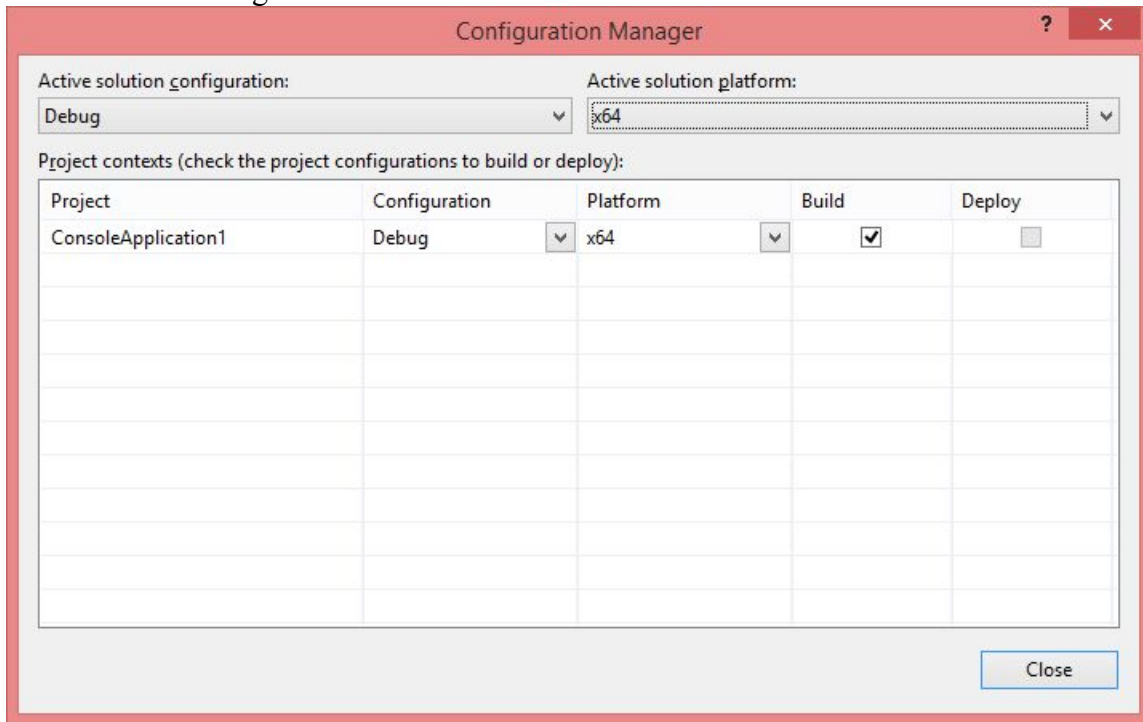


choose x64 and leave “Copy settings from:” and “Create new project platforms”:





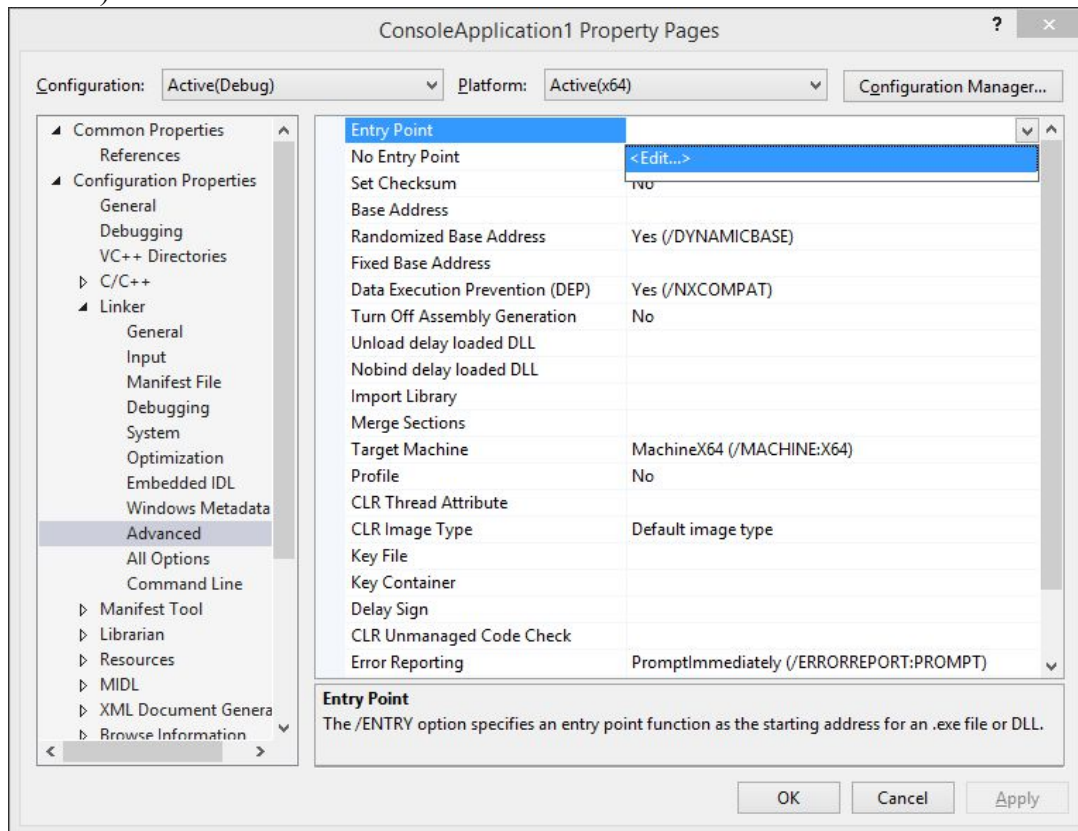
Your current configuration should look as below:



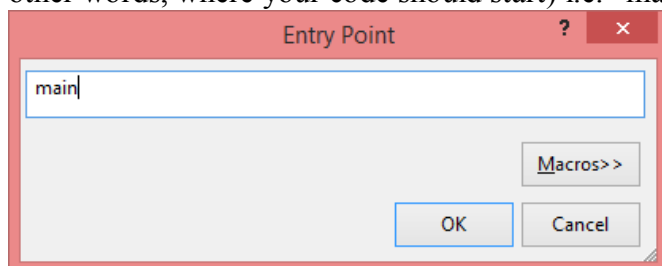
Click “Close” to accept configuration.



- f) The linker is aware of the entry point, so you need provide the name (the ‘main’ function). You have to inform it to be able to run the project. To do this, click [PROJECT>>Properties] from the VS menu then expand “Configuration Properties>Linker” and click on “Advanced” section. Then open “Entry Point” position, click on “Edit” and write the name of the main function (here just “main”):



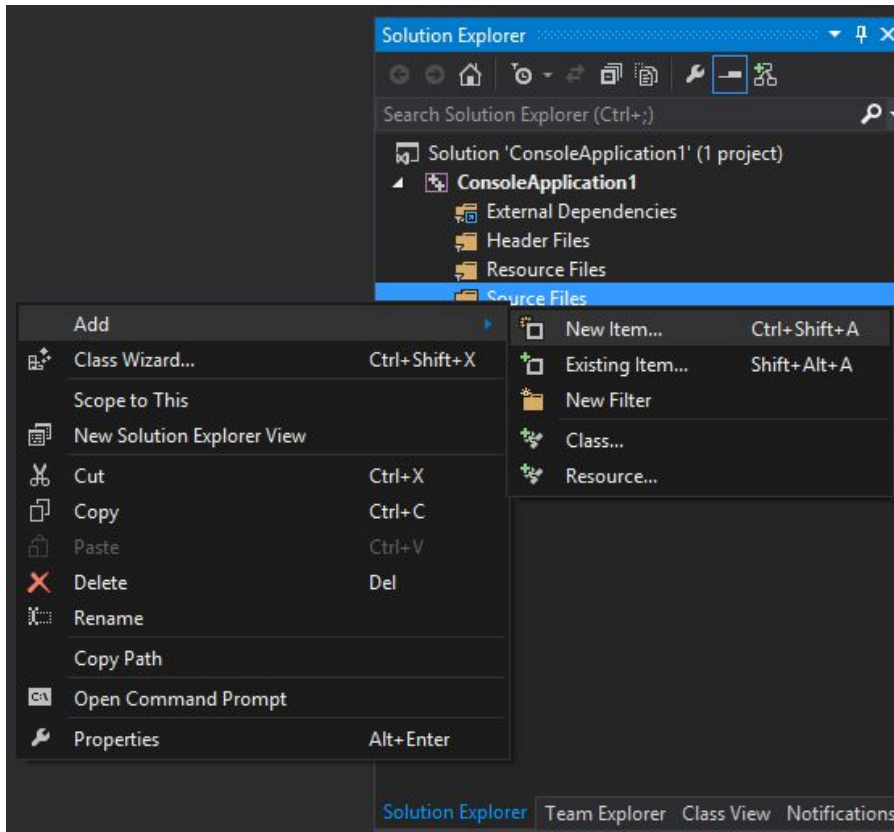
edit the value providing your ‘main’ function name chosen (an entry point for your application, in other words, where your code should start) i.e. “main”:



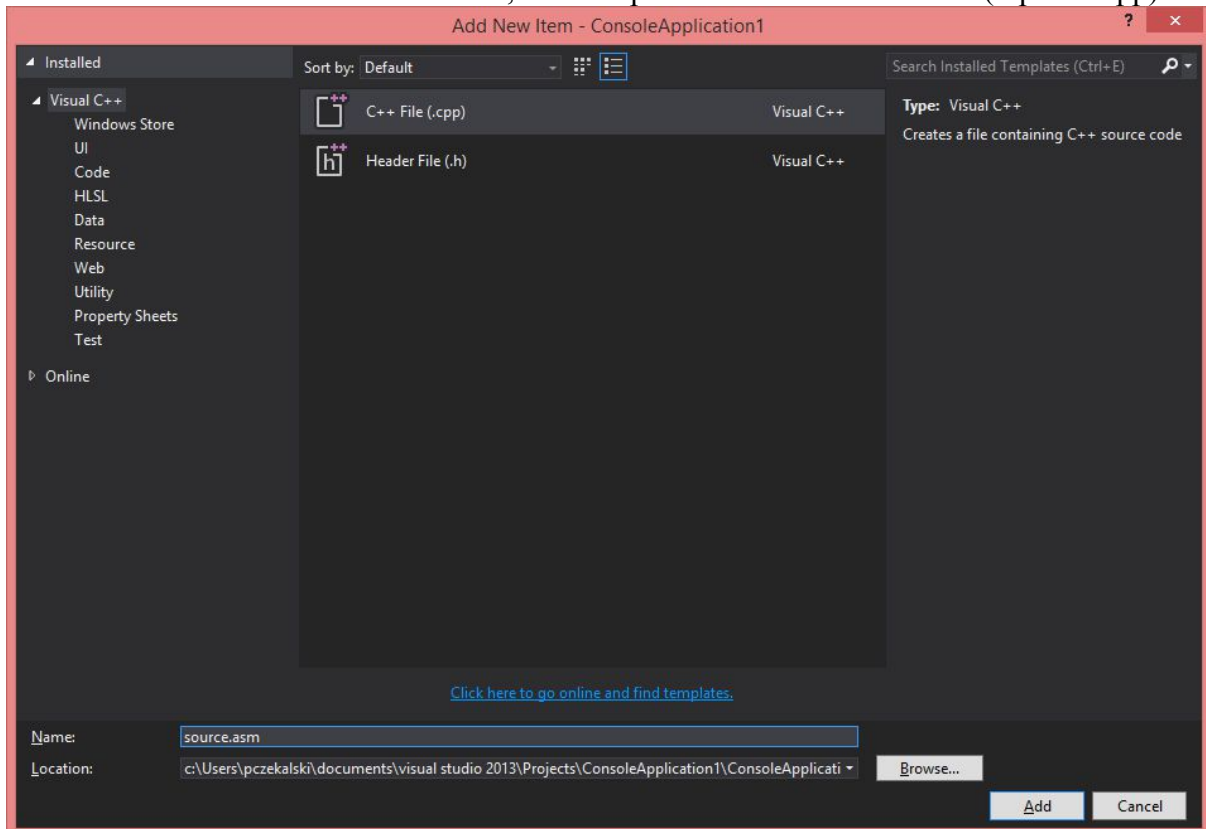
then click accept the settings (OK twice on the dialogs as above).



- g) Add your x64 assembler source file. To do so, right click on Solution Explorer “Source Files” the “Add” and the “New item”:



Choose C++ file in Visual C++ section, mind to provide .asm file extension (replace .cpp):



and click “Add” to accept.



h) Write some simple source file, compile and Run.

Mind the non-leaf function (a function that calls another function which is 'main' in this case) need to reserve enough space on the stack to handle at least 4 arguments (parameters), even if the number of parameters is less than 4, but also requires to round up the alignment of the stack reservation to the 16-byte memory boundary. If insufficient stack space is reserved or the 16-byte alignment is not satisfied, your application may crash.

```
extrn MessageBoxA : proc
extrn ExitProcess : proc
extern SetConsoleTitleA : proc
extern WriteConsoleA : proc
extrn GetStdHandle : proc

STD_OUTPUT_HANDLE = -11

.data
handle dq ?
MsBoxCaption db "Sample dialogbox header",0;
MsBoxText db "Hello world",0;
ConsoleText db "Hello, world!", 0;
ConsoleTextLen = $-ConsoleText;
Text db "This is title",0;
NumCharsWritten dw ?

.code

main proc
    sub rsp, 28h

    mov ecx, STD_OUTPUT_HANDLE
    call GetStdHandle
    mov [handle], rax

    lea rcx,Text
    call SetConsoleTitleA

    xor r9d,r9d
    lea r8, MsBoxCaption;
    lea rdx, MsBoxText;
    xor ecx,ecx
    call MessageBoxA

    mov rcx, [handle]
    lea rdx, ConsoleText
    mov r8d, ConsoleTextLen
    mov r9w, NumCharsWritten
    call WriteConsoleA
    mov rcx, 0;
```



```
    call ExitProcess
    add rsp,28h
main endp
end
```

i) After Your output should present similar communicate:

```
1>----- Rebuild All started: Project: lab8, Configuration: Debug x64 -----
1> Assembling Source.asm...
1> lab8.vcxproj -> lab8.exe
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
```



- j) Set the breakpoint on the beginning of the 'main' function i.e. on the line `sub rsp, 28h`:

```
Source.asm  X
ExitProcess PROTO
MessageBoxA PROTO

SetConsoleTitleA PROTO

.data

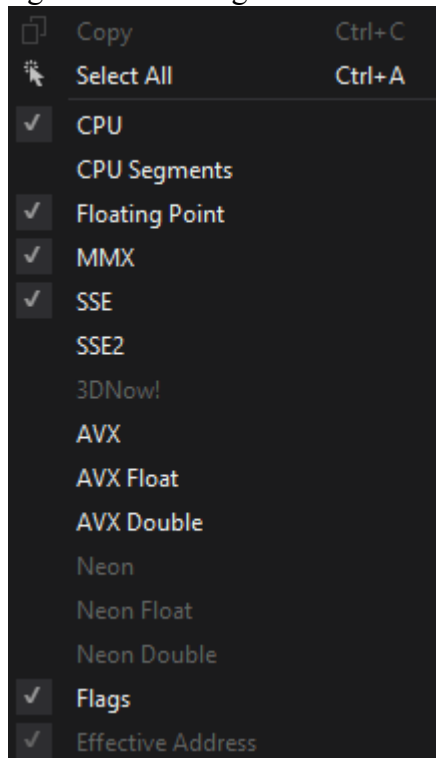
qword1 qword 12345678ffeeddcch;
MsBoxCaption db "Sample dialogbox header",0;
MsBoxText db "Hello world",0;
Text db "This is title",0;

.code

main proc
  sub rsp, 28h
  lea rcx,Text
  call SetConsoleTitleA
  xor r9d,r9d
  lea r8, MsBoxCaption;
  lea rdx, MsBoxText;
  xor ecx,ecx
  call MessageBoxA
  mov rcx, 0;
  call ExitProcess
main endp
end
```



- k) Start debugging (click F5), once the code execution stops on the breakpoint enable Registers window to observe the changes while debugging (click [DEBUG]>>Windows>>Registers) then right-click the Registers window and select options to enable Flags, MMX, SSE etc.:



- l) Debug your code.



4. During labs – tasks to compile:

1. Get ready your solution, compile and run sample program.
2. Debug and observe memory and registers. To start debugging press F10, to open the Registers and Memory Windows click “[Debug>>Windows]” in the VS menu when in Debug mode.
3. Modify the code according to the tasks given by leader, particularly extend the code by some simple Windows API functions.