Table of Contents

1. Programming in Assembler Laboratory 2
1.1. Laboratory 1 2
1.2. Introduction 2
1.2.1. Functions and functions calling from assembler 2
1.2.1.1. Function calling convention 3
1.2.2. Practice: Creating a C++ project and conversion to ASM 4
1.2.2.1. Creating a source code file 5
1.2.2.2. Converting a project from C++ to ASM 6
1.2.2.3. Creating a starting point and completing libraries 7
1.2.2.4. Marking an ASM file for compilation 9
1.2.3. Practice: Task to be performed 10
1.2.3.1. Code fragments to use 10
1.2.4. Tasks to be done for the report 14
1.2.5. Method of results verification 14

1.Programming in Assembler Labora

1.1Laboratory 1

Conû guration of the Visual Studio for simple assembler application wr

1.2Introduction

In this exercise, you will create a console application in pure assembl
64 bits) that will interact with the Windows operating system (Windows
functions via the Windows API.

This is just an example, creating complex applications with advanced
possible although rather pointless. On the other hand, it can be very u
computational services, tools, etc., which require compact code and e
speed. Note that the examples given are not limited to user interface
Windows API.

Since there is no ready-made project model for pure assembly applica
use the C++ console application model and adjust it for pure assemble
C++ code.

This approach requires manual reconû guration of several parameters i
particular:

- changing the compiler to ml64.exe;
- changing the target platform to x64;
- providing the name of the main function as the entry point ([Entry
system knows which function to call when the application is launch
languages and environments, this is the main() function by default, but in the case
assembler, there is no such rule and you have to specify which fun
- supplementing missing libraries: since version 2019 of the Visual
standard functions previously available by default in libraries link
disabled and must be explicitly indicated.

> Note! Because we are using which Visual Studio does not nativ
> as source code û les, they are not automatically saved when
> likecpp, .csû les. Please remember to always click save the
> compiling.

1.2.1Functions and functions calling from assembler

When writing code in assembler, each byte must be allocated, declare

manner, especially in the context of calling Windows API functions. Since we are working in an operating system, it is necessary to follow known as calling convention.

1.2.1.1 Function calling convention

We pass function parameters via registers, as long as their number do additional parameter we pass through the stack (ûfth and following pa mandatory to allocate space on the stack for arguments even if we pas called shadow space.[1]

1.2.1.1.1 Passing integer arguments

When passing integer numeric arguments to a function, in particular a following registers should be used in the order: - argument 1 !' RCX, - 3 !' R8, - argument 4 !' R9, - each subsequent argument is placed on th data alignment (for performance reasons). Sample C++ code:

```
func1(int a, int b, int c, int d, int e, int f);
// a in RCX, b in RDX, c in R8, d in R9, f, and then e, are pushed onto
stack
```

1.2.1.1.2 Passing ûoating-point arguments

When passing ûoating-point arguments, we load subsequent values in !' XMM0, - argument 2 !' XMM1, - argument 3 !' XMM2, - argument 4 !' XM parameter, similarly as for ûxed-point, through the stack. Sample C++

```
func2(float a, double b, float c, double d, float e, float f);
// a is XMM0, b in XMM1, c in XMM2, d in XMM3, f and then e are pus
the stack
```

1.2.1.1.3 Passing mixed arguments

When mixing ûoating point and decimal arguments, the positional syst type, so if we pass, for example, int1, ûoat1, ûoat2, int2, the values order): - int1 !' RCX, - ûoat1 !' XMM1, - ûoat2 !' XMM2, - int2 !' R9, - ûft stack. Sample C++ code:

```
func3(int a, float b, float c, int d, int e, float f); // a in RCX, b in
XMM1, c in XMM2, d in R9, f, then e, via the stack
```



Registers values are changed after a function call, so you sh a function call, registers RCX, RDX, R8, R9, XMM0-XMM3 wil

before the function call

1.2.1.1.4 Returning a value from a function

An integer value (including a pointer) is returned in register RAX. Floating-point values are returned in register XMM0.

1.2.1.1.5 Volatile and non-volatile registers

Some registers must remain unchanged upon return from the
their contents at the beginning, right after the function ca
Other registers can be changed freely as it will not a
operating system (volatile).
If it is necessary to use this first group, the contents of t
onto the stack or saved in memory and restored before exe
ed instructi

Registers that can be changed with impunity inside the function, witho
value, are (calling conventions):

- RAX, RCX, RDX, R8, R9, R10, R11, and XMM0-XMM5,
- additionally in processors that are equipped with YMM registers, t
  YMM15 registers can also be changed without having to restore th
- and in processors equipped with functions AVX512VL, all registers
  XMM31.

Registers that must not be changed (or rather must be restored to the
the function) are:

- RBX, RBP, RDI, RSI, RSP, R12, R13, R14, R15 and XMM6-XMM15.

1.2.2 Practice: Creating a C++ project and conversion to ASM

Create a new project: a command-line (console) application (Console Application .NET Core) in C++, n



Fig. 1: Creating a C++ project, native code

1.2.2. Creating a source code file

Delete the generated .cpp file (from VS) and then add a new .asm file to the project (not the solution!) and selecting to add a new element (Fig. 2), and change its name and extension (Fig. 3)

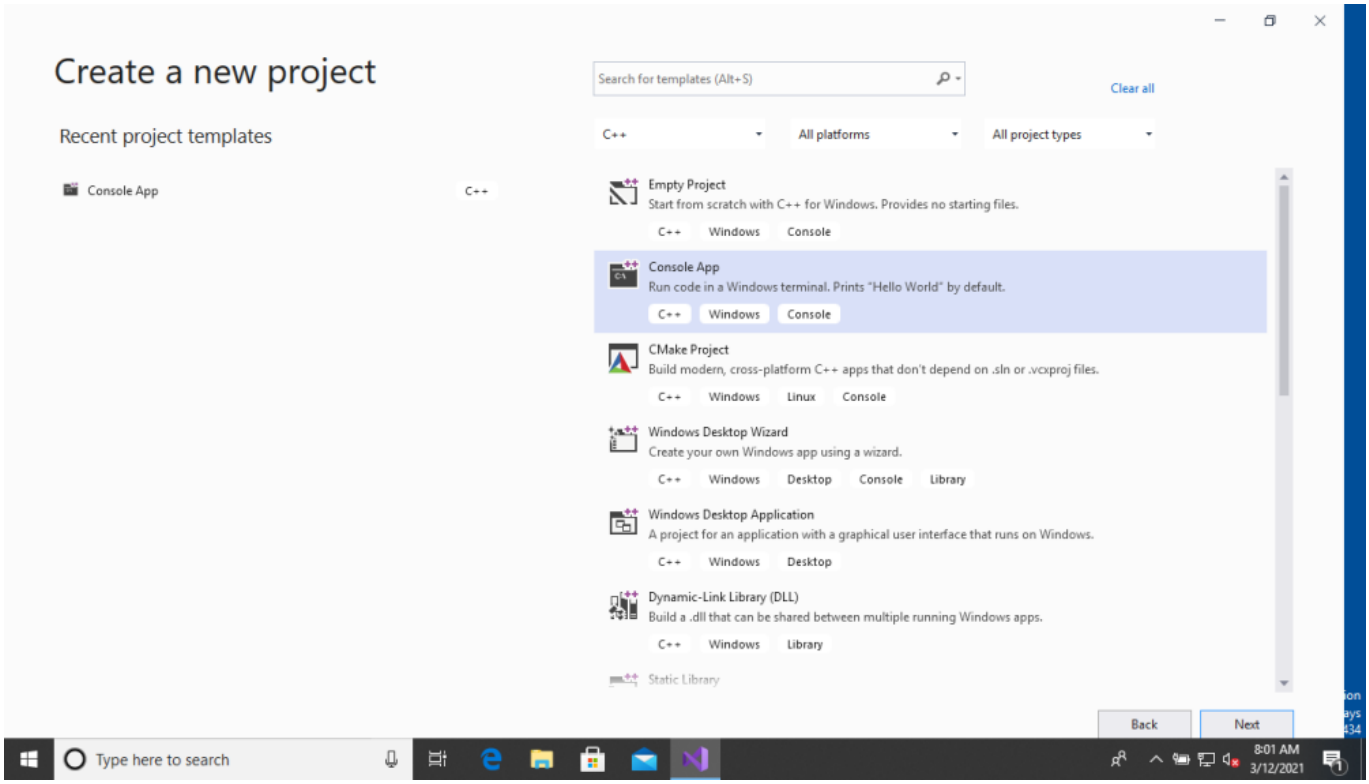


Fig. 2: Adding a new element

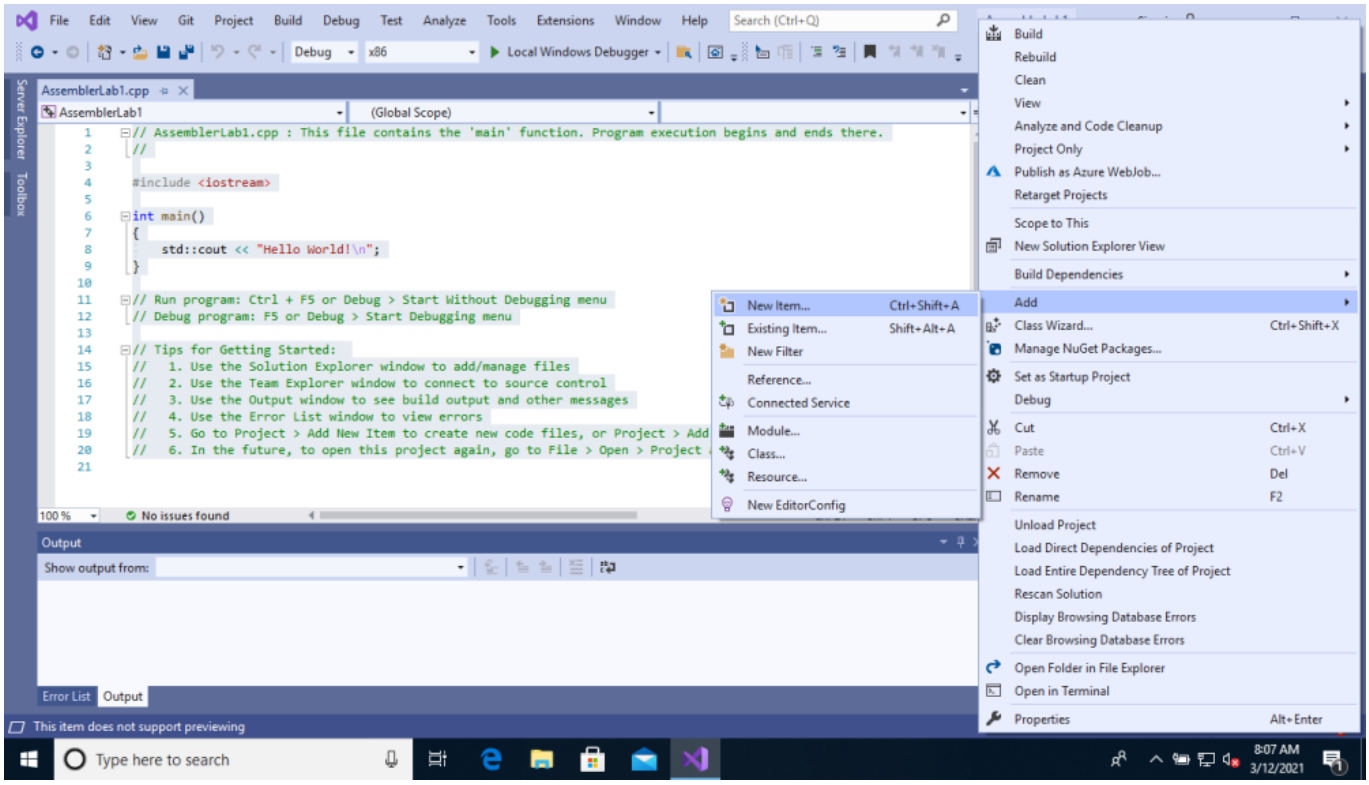


Fig. 3: Adding a new element - asm file

1.2.2. Converting a project from C++ to ASM

To convert, you need to tell VS that the source files will be compiled u
compiler. To do this, right-click on the project and select the option t
figure 4. In the settings, as well as targets, .props (figure 5)

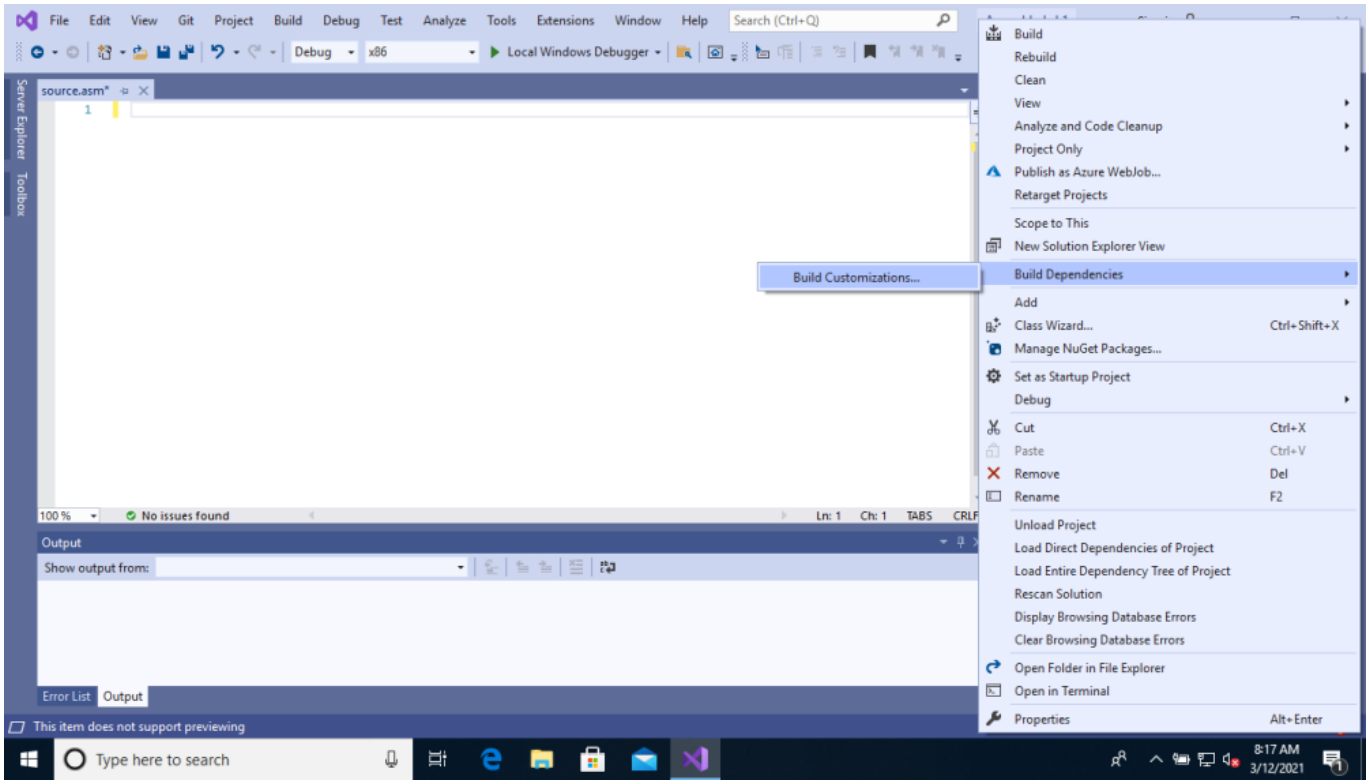


Fig. 4: Choosing the project build customization menu

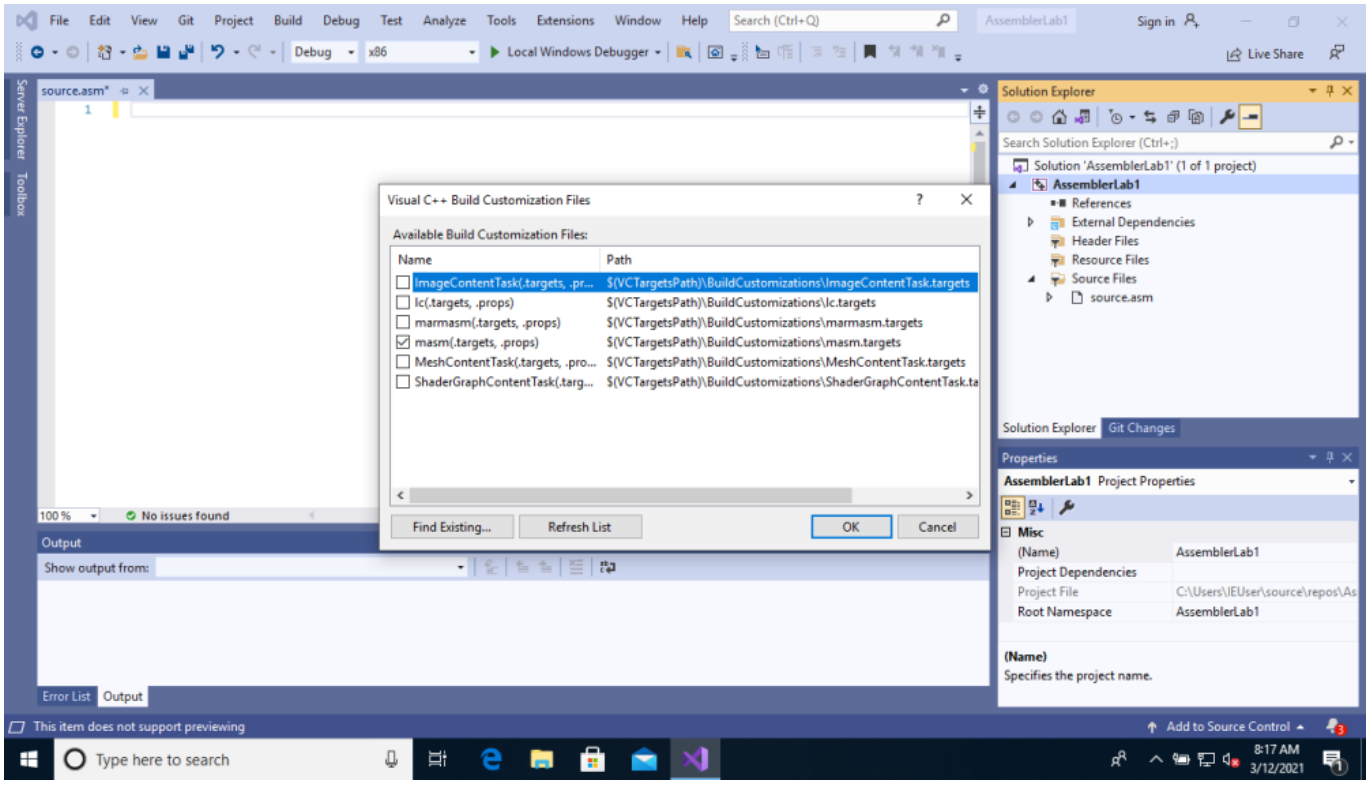


Fig. 5: Choosing the MASM compiler

Next, change the type of application (32-bit build platform) to x86 (32-bit application): Build! Configuration Manager menu figure 6

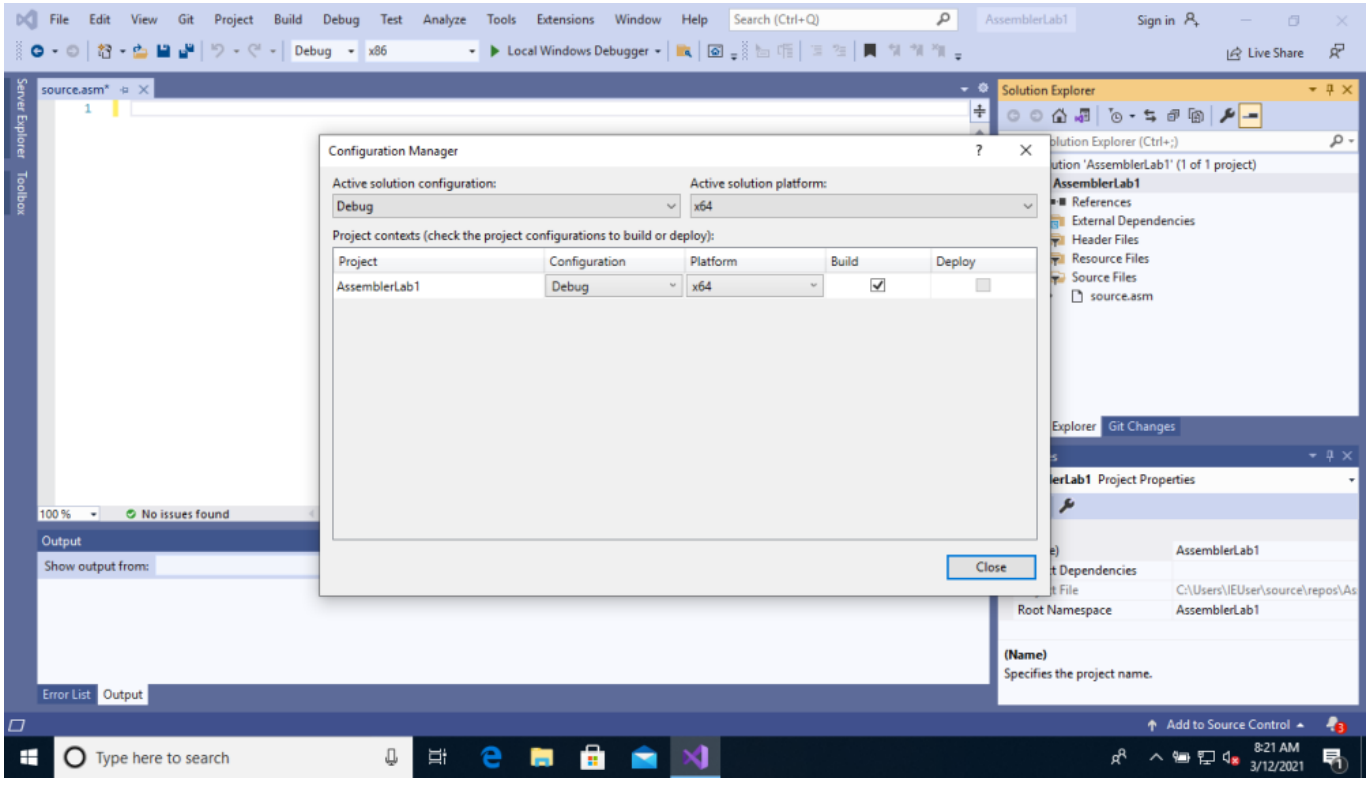


Fig. 6: Environment configuration - building the 64-bit version

1.2.2. Creating a starting point and completing libraries

In this step, we will inform the linker what starting point to enter into languages, this main() function by default. In the case of an assembler, t

assumption, so the source code must be consistent with the value set

> The name of the code entry point function given to the linke
> code. In case of discrepancies, the program will work incorr
> throw an exception at startup!

In the given example, the name of the function is set to the conû guration window, you should select the project (not the solution!) in the projec project, select Properties, or use the shortcut [Alt+Enter]. A window will appe Then in Linker section, select Advanced option and in Entry Point û eld enter the name of the function. In our case, it will be:

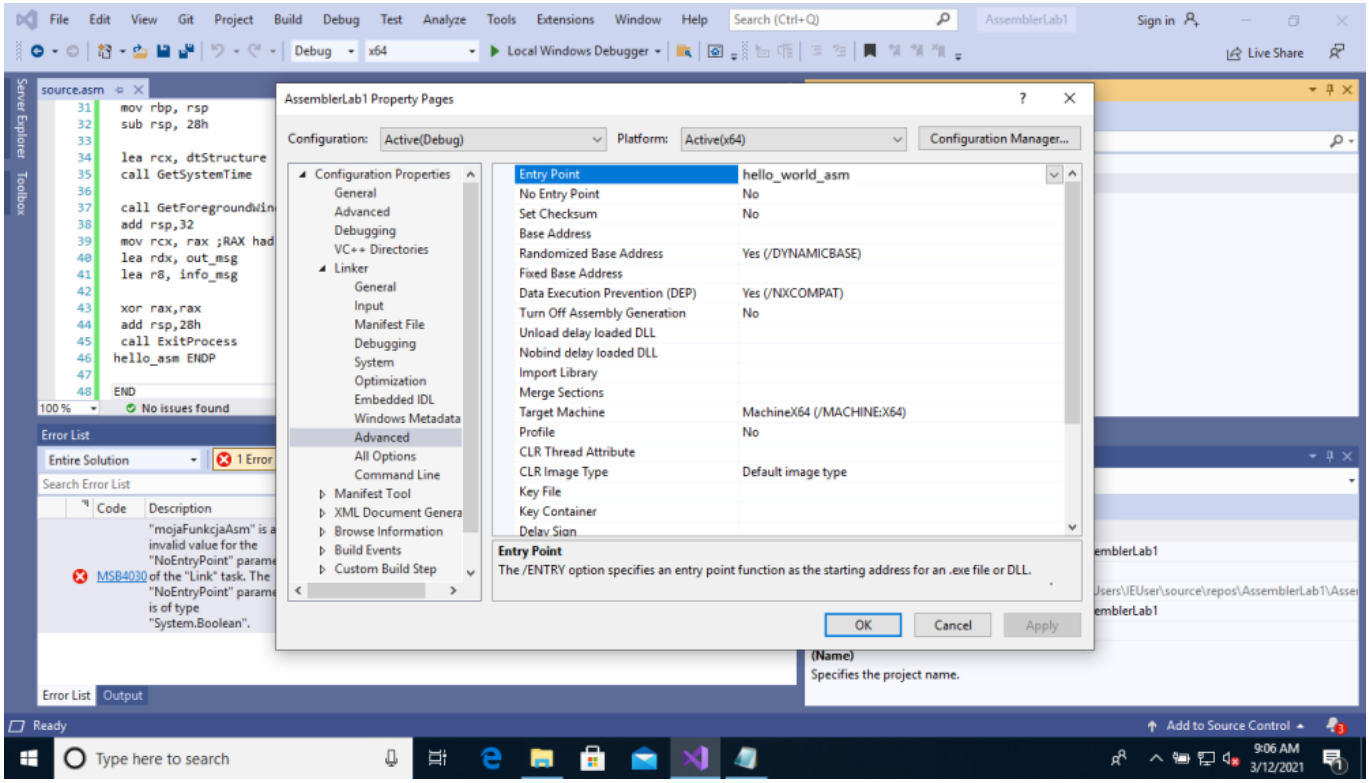


Fig. 7 Conû guration of the program entry point

In this section, you should also add the missing libraries that we will Windows version and current update level, this step may not be neces particular, these libraries contain printf functions such as
Note - the given libraries should be added to those that are already th libraries located there (kernel32.lib, user32.lib) will result in the inability to standard dialog boxes and interact with the Windows API. To add libra the Linker menu, then Additional Dependencies and select <Edit...> option after expanding the selection list. Then, in the upper û eld, enter the names additionally available during compilation (one item per line).
Libraries to be added are:

- legacy_stdio_definitions.lib
- legacy_stdio_wide_specifiers.lib

Details in Fig. 8.

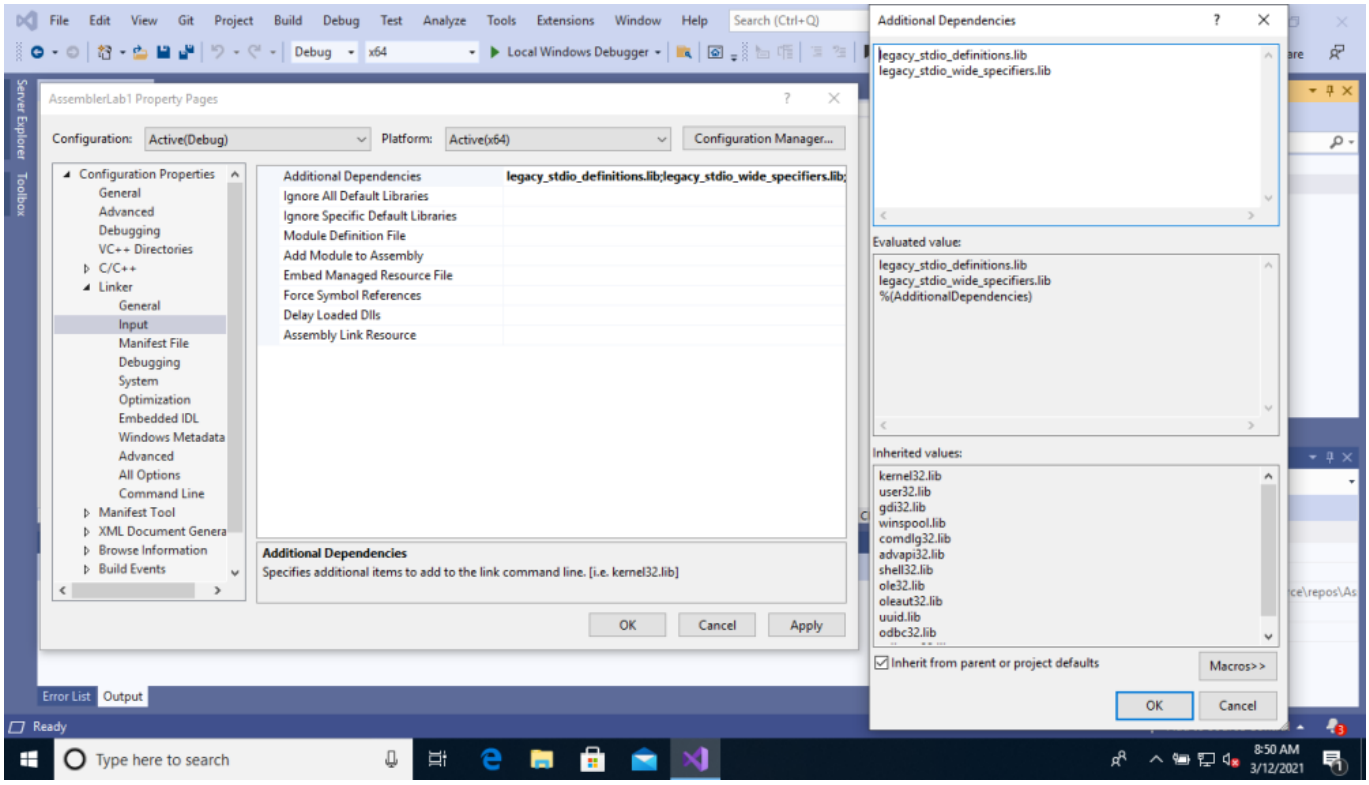


Fig. 8: Configuring additional libraries

1.2.2. Marking an ASM file for compilation

In some cases, an ASM file is not automatically recognized as a source file with assembler code. Therefore, you must explicitly inform the environment about this. To do this, in the project tree panel, select the file, then in the properties section, select Item Type and set its value to Microsoft Macro Assembler. Details in Figure 9.

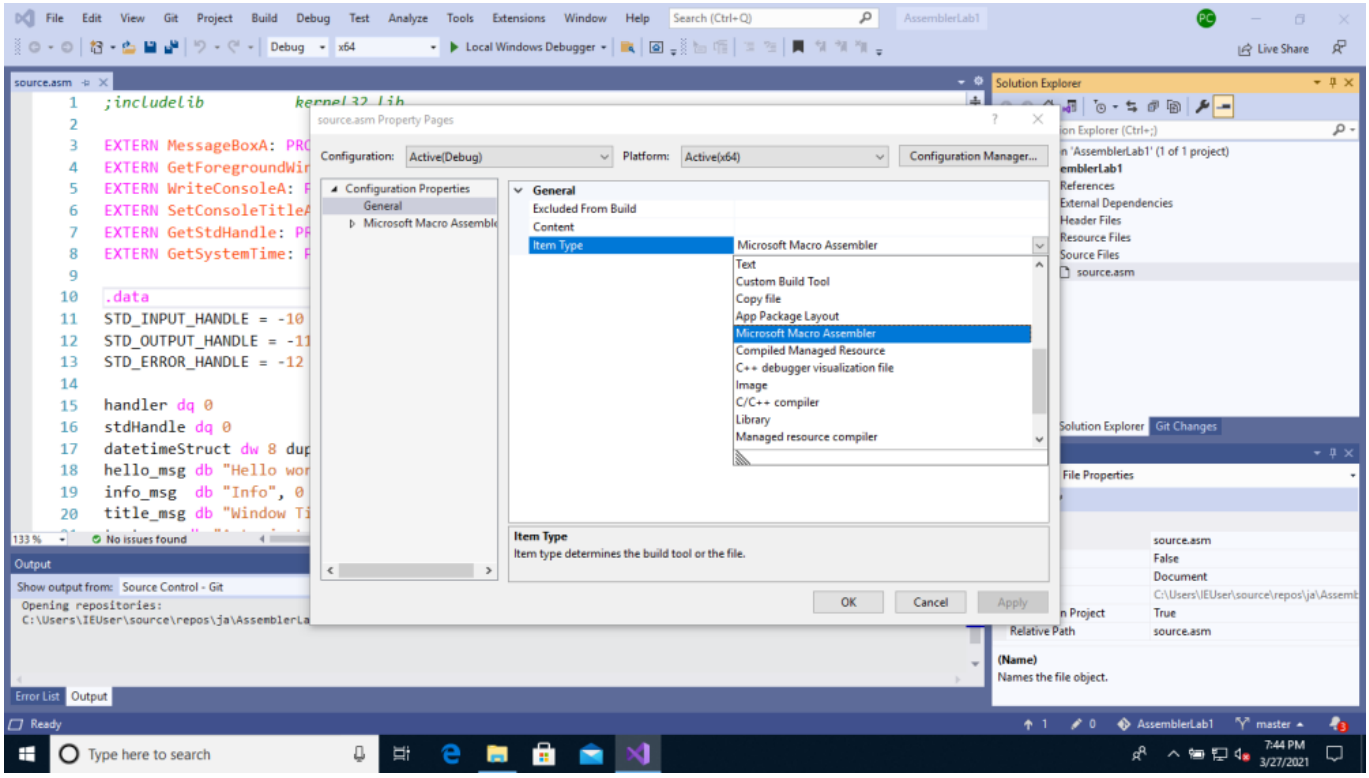


Fig. 9: Marking an ASM file as a source file with assembler code

1.2.3 Practice: Task to be performed

Implement reading of the current date (System time) [2] and displaying the structure SYSTEMTIME [3]. The structure consists of 8 16-bit values.

> Note - do not declare the structure as such, just for simplic
> array of 8 Word (16bit) values in the data section.

Convert the year, month and day to ASCII text form and display on the

- in the command line (cmd) using the WriteConsoleA function;
- in the dialog box, using the MessageBoxA function;

> Pay attention to the position of individual words in
> corresponding values.

You can also display the current hour, minutes and seconds.

1.2.3.1 Code fragments to use

Remember to declare the code and data sections using the directives. At the en of the code, please include the END directive.

> Code and Data sections historically were named segments.
> names in some sources.

1.2.3.1.1 Data section

Each text string must end with a byte with binary value 0. It is named use international characters because we use the ASCII character set (A sample data segment that was used in the presentation looks like th

```
...

.data
STD_INPUT_HANDLE = -10
STD_OUTPUT_HANDLE = -11
STD_ERROR_HANDLE = -12

handler dq 0
stdHandle dq 0
datetimeStruct dw 16 dup (0)
```

```
hello_msg db "Hello world", 0
info_msg db "Info", 0
title_msg db "Window Title JA",0
text_msg db "And this is a very long text to display in the console!",1
dummy dq 0

...
```

To use Windows API functions, you have to tell the assembler how the
match the names from your code to those in the library.
To do this, you need EXTERN and PROC directives:
EXTERN means that the functions are not defined in your source code
- here libraries. PROC specifies that the symbols represent the names

> In assembler there is no difference between function and proce
> is named procedure.

```
...

includelib kernel32.lib

EXTERN MessageBoxA: PROC
EXTERN GetForegroundWindow: PROC
EXTERN WriteConsoleA: PROC
EXTERN SetConsoleTitleA: PROC
EXTERN GetStdHandle: PROC
EXTERN GetSystemTime: PROC

...
```

We also usually include information about the library.

1.2.3.1.2 Code section

After running the program, we need to save the values of certain reg
later. At the very end, you should also call a function that ends the pr
return code to the operating system.

The minimal set of instructions looks like this:

```
...
.code
hello_world_asm PROC
push rbp ; save frame pointer
mov rbp, rsp ; fix stack pointer
sub rsp, 8 * (4 + 2)
```

```
.... ; your program code here

mov rsp, rbp
pop rbp
ret
hello_world_asm ENDP
END
```

Note, in the above example, the main_proc_asm was indicated as the [Entry Point] to the program, in the hello_world_asm project. Adjust to your naming.

As you can see, the whole thing comes down to reserving and putting called shadow space, i.e. for saving the values of registers and rest finished executing.

In the program, we use interaction with graphic elements such as the Window) and the data stream (STD_OUT). When calling the function, it which graphic object we want to interact with or create a child object To do this, we need to get two handles:

- a handle to the current main application window at the operating s Foreground Window)
- a handle to a terminal-type data stream, into which information ca

Note! To display something on the console, you should a virtually. Therefore, any string of characters redirected to ST console as long as there is a special code at the end indica These are two numeric values, in the order 13 and 10. Henc data segment, after the content, 13, 10 there is. If you don't do this, you will go to the window and manually press ENTER to display redirected there from your program. Otherwise, the window wil text will not appear!

To read the handles, two functions from the Windows API will be usefu (handles) to the appropriate objects. These values should be saved a

Reading the handle to the main window, this handle should be used wh of the Message Box type (and any other):

```
...
call GetForegroundWindow
mov handler, rax
...
```

In the above example, the value returned by the function (via the RAX variable handler of the DQ type (64 bits, i.e. simply an address in memory

To read the handle to the inside of the terminal, you need to use anot

```
...
mov rcx, STD_OUTPUT_HANDLE
call GetStdHandle
mov stdHandle, rax
...
```

However, this function requires one argument, which data stream we w
possibilities:

- STD_INPUT_HANDLE = -10 - standard input - it can be used to rea
- STD_OUTPUT_HANDLE = -11 - standard output - it is used to write
  remember the ending ENTER, i.e. 13,10;
- STD_ERROR_HANDLE = -12 - stream for errors;

An example code that displays the text in the variable text_msg (at the address is
below. The first parameter is the console handle read above. In the th
specify how many characters should be displayed. The fourth paramete
function will return information about how many characters it actually
what this is for).

```
...
; WINAPI WriteConsoleA(
; RCX => _In_ HANDLE hConsoleOutput,
; RDX => _In_ const VOID *lpBuffer,
; R8 => _In_ DWORD nNumberOfCharsToWrite,
; R9 => _Out_opt_ LPDWORD lpNumberOfCharsWritten,
mov rcx, stdHandle
lea rdx, text_msg
mov r8, sizeof text_msg
mov r9, dummy
call WriteConsoleA
...
```

To display a dialogue box, use a pointer to the main application windo
this order are a handle, two pointers, to texts (they must end with a z
content that the dialogue box will display, and R8 is a pointer to the w
to the window) that are to appear in header and dialogue body and a na
which buttons and icons to use in the dialogue box[4].

```
...
; WINUSERAPI int WINAPI MessageBoxA(
; RCX => _In_opt_ HWND hWnd,
; RDX => _In_opt_ LPCSTR lpText,
; R8 => _In_opt_ LPCSTR lpCaption,
; R9 => _In_ UINT uType);
mov rcx, handler
mov rdx, offset hello_msg
mov r8, offset info_msg
mov r9, 0 ; MB_OK
```

```
and rsp, not 8
call MessageBoxA
...
```

Note - in the above case, the stack was explicitly aligned with the instruction and (not). It is a requirement of the x64 calling convention. If you call the system function you sho stack pointer so that it is divisible by 16.

1.2.4 Tasks to be done for the report

Let's start wth debugging. If the program runs correctly please set the assembly code. Open register window and observe the content of regis execution. Please do not step into Windows functions including WriteC use step over instead. In the report answer the following questions.

- RIP - note how its content diû ers with every instruction in the seri type of instruction executed? Compare changes with the disassemb of bytes instructions occupy.
- While calling WriteConsoleA look at the memory dump window at th What is there in the memory? You can try diû erent view modes.
- While calling MessageBoxA function look at the memory dump wind RDX and in R8. What is there in the memory? You can try diû erent
- Include in the program the instruction which sets the RAX to some before calling previously mentioned functions. Does the RAX regis returning from these functions?
- Try to replace the instruction mov rdx, oû set hello_msg with lea proper argument). Explain if they are identical or not.

1.2.5 Method of results veriû cation

The texts and windows that appear on the screen should correspond to attention to whether random strings of characters are generated, whic about the ending 0.

1) https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?vie
2) https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysi
3) https://docs.microsoft.com/en-us/windows/win32/api/minwinbase/ns-mi
4) https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuse

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - RAU6 Dokuwiki

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s1:apl:lab:start&rev=17326

Last update: 2024/11/26 13:29