

## 32-bit Assembler is Easy

Contrary to popular legend, if you can write a Windows application in a compiler based language ( C/C++, Visual Basic etc), you can also write it in Assembly, especially if you choose a Macro Assembler such as MASM.

32-bit Assembler is Easy, why and how to develop using the assembler? Start learning to program in Assembly now!

32-bit Assembler is Easy, why and how to develop using the assembler? Start learning to program in Assembly now!

### Introduction

Contrary to popular legend, if you can write a Windows application in a compiler based language ( C/C++, Visual Basic etc), you can also write it in Assembly, especially if you choose a Macro Assembler such as MASM.

### Why develop in assembler?

1. When written correctly, assembly language produces high speed, reduced final size executables that are beyond the capacity of the best compilers.

2. Using an assembler, you can write dynamic link libraries with the above mentioned advantages, that can be used from any other language you use ( such as C/C++, VB etc )that can call a DLL.

3. Similarly, writing a static library, you are able to directly link them into your programs written in other languages since it will be in the object module format that is used by the Visual C++ compiler.

### Bare Bone Executable

.386

Tells the assembler to use the 80386 instruction set. This is the safest bet unless you want to use instructions available only on later processors such as .486, .586 etc.

.MODEL FLAT, STDCALL

.MODEL specifies the memory model of the program. Only one model is applicable under Win32 no matter what type of application you develop

(standard executable, dll, static library, console etc), the FLAT one. That is to say, we need not be concerned with memory model or segments like we used to do the old days of the 16-bit world. The memory is a large continuous space of 4 GB and hence, you don't have to play with segment registers.

STDCALL informs the assembler how the parameters are passed to the functions. You don't need to bother about this in your first steps.

#### OPTION CASEMAP:NONE

Function names etc are case sensitive. There are more options but you do not need to bother for them now.

#### Include ...

This directive tells the assembler to open the file the name of which follows and place its contents there. In our case we used WINDOWS.INC, kernel32.inc, user32.inc since our program needs definitions of constants/structures and function prototypes that reside there.

#### IncludeLib ...

The easiest way to tell the assembler which import libraries your program uses. In our case, we need user32.lib and kernel32.lib.

#### .DATA

A section that contains initialized data of your program (variables). In our example, two (multi)BYTE variables are defined, namely MsgCaption and MsgBoxText. Something like string variables in the High level languages.

The .DATA? can be used for uninitialized data and the .CONST section contains declaration of constants (can never be modified in your program).

You don't have to use all three sections in your program. Declare only the section(s) you want to use.

#### .CODE

This is where your code resides

Start:

;your code here

End Start

where Start is any arbitrary label used to specify the extent of your code. Both labels must be identical.

The above could be considered as a skeleton project that is only missing the actual code. In our example we have two lines of code with Windows API function calls:

```
Invoke MessageBox, NULL, Offset MsgBoxText, Offset MsgCaption, MB_OK
```

```
Invoke ExitProcess, NULL
```

You can assemble our sample code with the MASM assembler using:

```
ml /c /coff /Cp assembleriseasy.asm
```

and link with:

```
link /SUBSYSTEM:WINDOWS /RELEASE /VERSION:4.0 assembleriseasy.obj
```

Please note that instead of using command line, there are a lot of tools most (if not all) of which are free, to help you build your applications easily.

## Conclusion

I know I haven't gone into any detail of any assembler instructions but I did this on purpose. I just wanted you to see the basic structure of an assembly listing file to verify that it is simple enough to start programming in Assembly.

## Assembler Links

[WinAsm Studio IDE](#), [tutorials](#), [source code](#), [Assembler board](#)