32bit Assembly Quick Start using Visual Studio and MASM on Windows

Jim Weller

<jim atat jimweller doot net>

9/5/2003

Describes howto use nmake and MS Visual Studio to create, compile, and debug programs written in assembly language for masm on x86. Serves as a solid introduction to microsoft makefiles and assembly programming. It is intended to get developers with other programming experience up and running quickly. It is perfect as a first assembly lesson in a college computer architecture or assembly course.

Table of Contents

Legal Notices 1
License 1
Notices and Acknowledgements 2
Introduction 2
Requirements 2
Creating your project 3
Compiling your project 12
Debugging your project 16
Conclusion 20
References 20

Legal Notices

License

Notices and Acknowledgements

Much of the code here is inspired by and drawn from "Assembly Language for Intel-Based Computers" by Kip Irvine

Windows, Visual Studio, Visual C++, and MASM are copyright Microsoft

Pentium, x86, and IA-32 are copyright Intel

Generally, copyrights, patents, and trademarks are owned by their owners.

Introduction

When I first took up assembly programming at the beginning of a computer (read intel) architecture class I couldn't find a hello world program that would compile as a 32 bit console application using the tools I had for windows; visual c++ and masm. Most references on the net were 16 bit code and none were with visual studio. This guide aims to fill that gap. If you find this useful please drop me an email so I can brag to my grandparents.

Assembly language is the programming language that is closest to the hardware (next to machine code, but that's not much of a "language"). It is often used in development tools like compilers, when tight controls or extra speed is necessary. It is also sometimes portrayed as arcane and inapproachable. But is a core course requirement in any respectable computer science program.

Sadly, most assembly books come with a copy of the MS assembler, MASM. This pretty much forces you into a Microsoft/intel paradigm of programming which is different from other common tools and syntaxes (NASM for example. Most assembly books also start you off nicely wrapped in a programmers framework; meaning you include and use procedures and macros before you ever know what they mean. A clean- room, from scratch approach, while difficult, is a more effective learning tool for me.

This quick start is targeted at computer students and professionals who are new to assembly programmning, but have experience in another high level language. It is a good supplement to a MASM based computer text book. A lay Windows user could probably walk through this guide with no experience what so ever. But programming background and experience with the command line, compilers, and IDE's will be helpful.

Requirements

There are three requirements to using the development framework suggested in this quick start:

- You need an intel x86 (or compatible) computer running a 32 bit windows (NT,2000,XP,et al). I'm not sure about 9x/ME.
- You need a copy of Visual C++ installed. I'm using version 6.0sp5 (pre-.NET). YMMV.
- You need a copy of MASM. I use version 6.1.5 in my examples. It's what came with my "Assembly Language for Intel-Based Computers" text book. YMMV.

I assume the following locations for masm, visual studio, and your project respectively. You'll need to interpolate as necessary

- c:\masm615
- C:\VS6

- %USERPROFILE%\Desktop\helloworld (example "C:\Documents and Settings\jim\desktop\helloworld")

Creating your project

This section guides you through setting up VC++ to work with assembly. You'll create a workspace and project. You'll add a .asm and .mak file to the project. When you've completed this section you'll be ready to compile an assembly program.

Start by opening Visual C++. Use FILE->NEW to create a new blank project called 'helloworld'. (see figure 1)

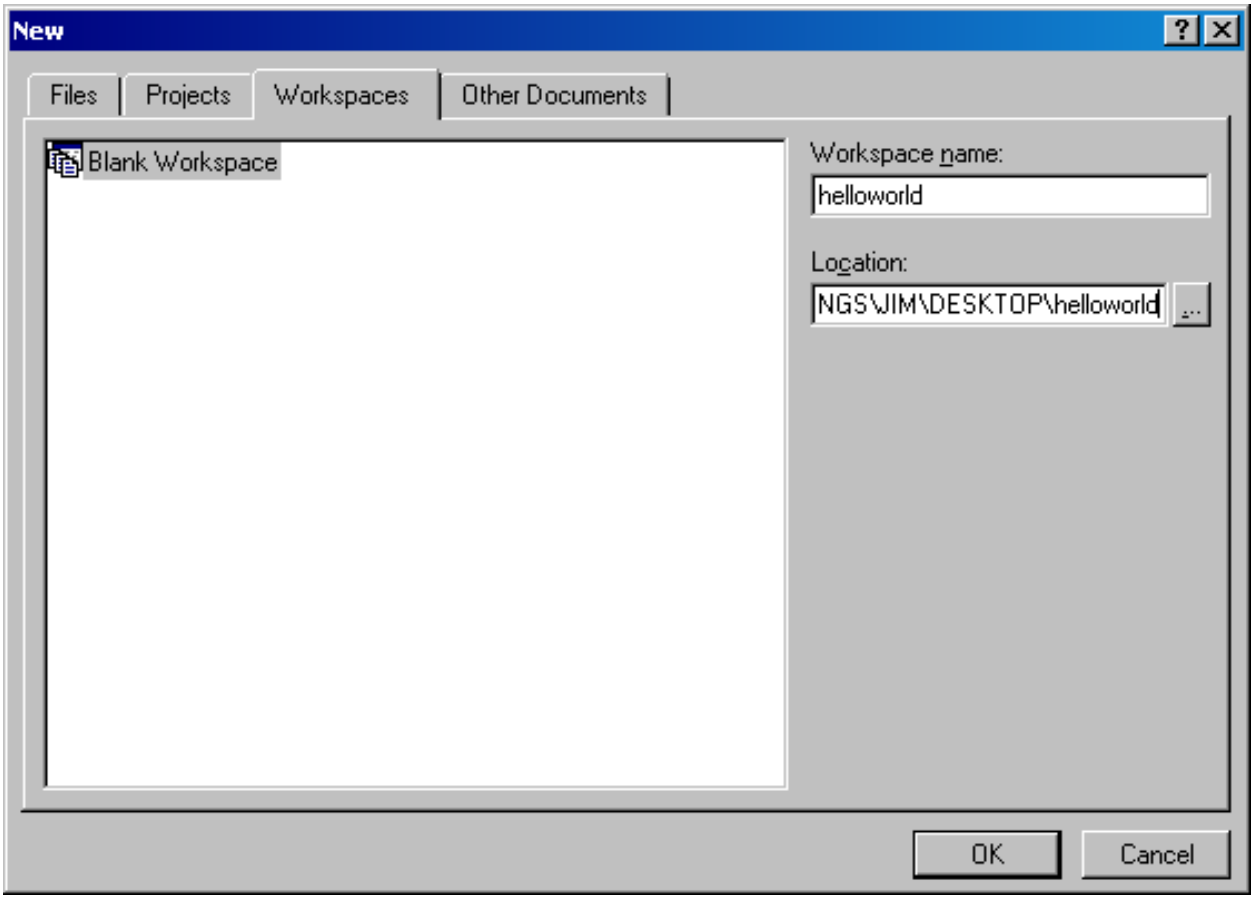

Figure 1

Now we'll add a make file project to this workspace. When the new workspace is open, right click it in the file view and select "Add New Project To Workspace..." (see figure 2)

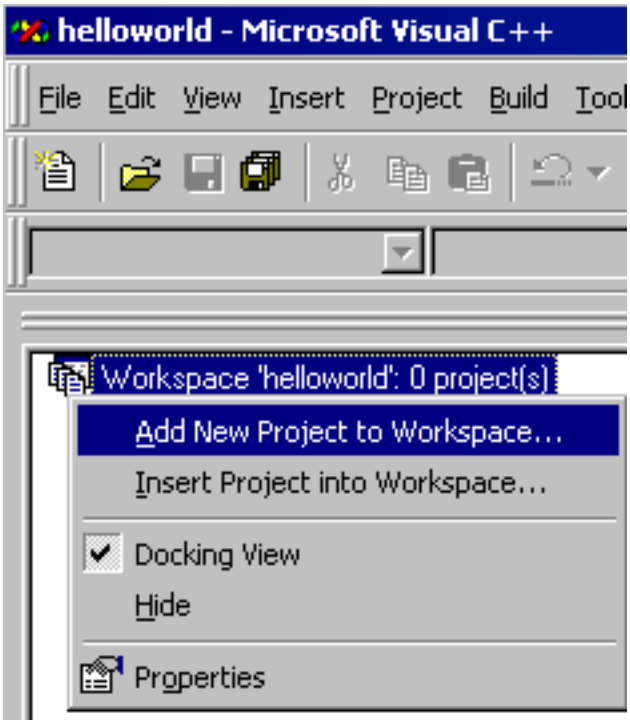

Figure 2

Choose "Makefile" and set the name to "helloworld". Use the same directory as for your workspace. If your skilled in VC++ do what you want. Change the radio buttons to read "Add to current workspace..." (see figure 3). Accept the defaults on the wizard that follows.

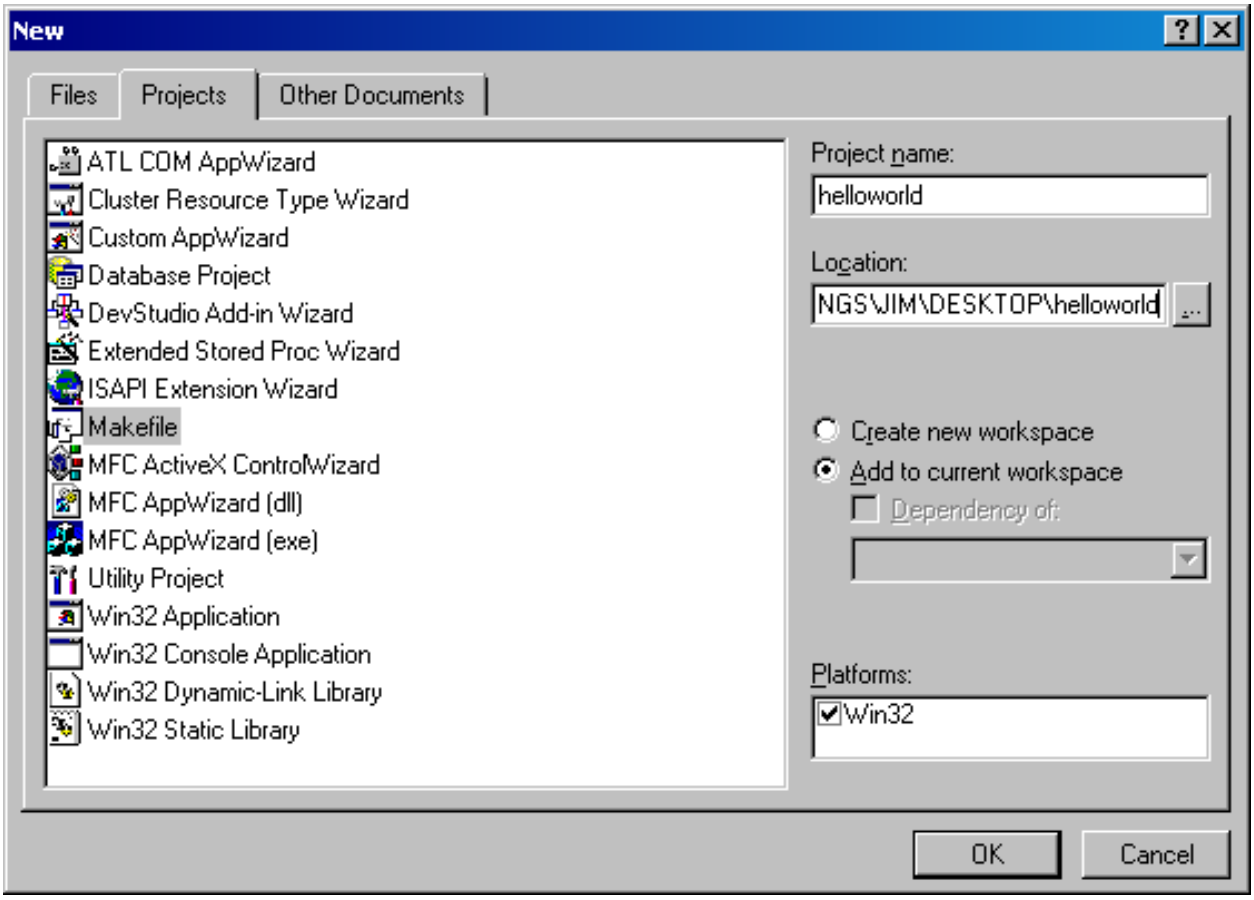

Figure 3

Before we go any further you'll need to have a sample assembly program file and a make file for your project. Code listings and links follow. Save these files to you Workspace/Project folder.

helloworld.asm [helloworld/helloworld.asm]

```
; This is a very simple 32 bit assembly program
; that uses the Win32 API to display hello world on the
; console.

TITLE Hello World in Win32 ASM       (helloworld.asm)

.386
.MODEL flat, stdcall
.STACK 4096

; ------------------------------------------------------------------------
; These are prototypes for functions that we use
; from the Microsoft library Kernel32.lib.
; ------------------------------------------------------------------------

; Win32 Console handle
STD_OUTPUT_HANDLE EQU -11               ; predefined Win API constant (magic)

GetStdHandle PROTO,                 ; get standard handle
        nStdHandle:DWORD                                ; type of console handle

WriteConsole EQU <WriteConsoleA>    ; alias
```

```
WriteConsole PROTO,                      ; write a buffer to the console
        handle:DWORD,                          ; output handle
        lpBuffer:PTR BYTE,                    ; pointer to buffer
        nNumberOfBytesToWrite:DWORD,            ; size of buffer
        lpNumberOfBytesWritten:PTR DWORD,  ; num bytes written
        lpReserved:DWORD                       ; (not used)

ExitProcess PROTO,                  ; exit program
        dwExitCode:DWORD                       ; return code

; ---------------------------------------------------------------------------




; ---------------------------------------------------------------------------
; global data
; ---------------------------------------------------------------------------

.data
consoleOutHandle dd ?                   ; DWORD: handle to standard output device
bytesWritten     dd ?                   ; DWORD: number of bytes written
message db "Hello World",13,10,0  ; BYTE: string, with \r, \n, \0 at the end

; ---------------------------------------------------------------------------



.code

; ---------------------------------------------------------------------------
procStrLength PROC USES edi,
        ptrString:PTR BYTE ; pointer to string
;
; walk the null terminated string at ptrString
; incrementing eax. The value in eax is the string length
;
; parameters: ptrString - a string pointer
; returns: EAX = length of string prtString
; ---------------------------------------------------------------------------
        mov edi,ptrString
        mov eax,0                       ; character count
L1:                                ; loop
        cmp byte ptr [edi],0      ; found the null end of string?
        je  L2                      ; yes: jump to L2 and return
        inc edi                      ; no: increment to next byte
        inc eax                      ; increment counter
        jmp L1                       ; next iteration of loop
L2: ret                           ; jump here to return
procStrLength ENDP
; ---------------------------------------------------------------------------



; ---------------------------------------------------------------------------
procWriteString proc
;
; Writes a null-terminated string pointed to by EDX to standard
; output using windows calls.
; ---------------------------------------------------------------------------
        pushad

        INVOKE procStrLength,edx             ; return length of string in EAX
        cld                             ; clear the direction flag
                                        ; must do this before WriteConsole

        INVOKE WriteConsole,
            consoleOutHandle,               ; console output handle
            edx,                    ; points to string
```

```
            eax,                    ; string length
            offset bytesWritten,            ; returns number of bytes written
            0

        popad
        ret
procWriteString endp
; -------------------------------------------------------------------------



; -------------------------------------------------------------------------
main PROC
;
; Main procedure. Just initializes stdout, dumps the string, and exits.
; -------------------------------------------------------------------------
        INVOKE GetStdHandle, STD_OUTPUT_HANDLE ; use Win32 to put
                                                ; stdout handle in EAX

        mov [consoleOutHandle],eax              ; Put the address of the handle in
                                                ; our variable

        mov edx,offset message                  ; load the address of the message
                                                ; into edx for procWriteString

        INVOKE procWriteString                  ; invoke our write string method.
                                                ; It'll check EDX

        INVOKE ExitProcess,0                    ; Windows method to quit

main ENDP
; -------------------------------------------------------------------------

END main
```

helloworld.mak [helloworld/helloworld.mak]

```
# A very simple make file for a windows 32 bit assembly console program
# it assembles and links

# nmake help is online at:
#  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcug98/html/_asug_macros_and_nma

# Assemble the code into coff format producing map and listing files,
# including symbolic debugging info. Try "ml /?" for more options
# and descriptions

# 32 bit link our .obj file with the kernel32.lib file and create an exe file

all: helloworld.exe

helloworld.exe: helloworld.asm
        ml /nologo /coff /c /Zi /Fl /Fm  $?
        link32 /nologo /DEBUG /incremental:no /subsystem:console /entry:main /out:debug\helloworld.e
```

The next step is to add downloaded files to the project. You'll add our helloworld.asm assembly program. You'll also add an nmake makefile, which is similar so gmake. Right click your project and click "add files.." (see figure 4)

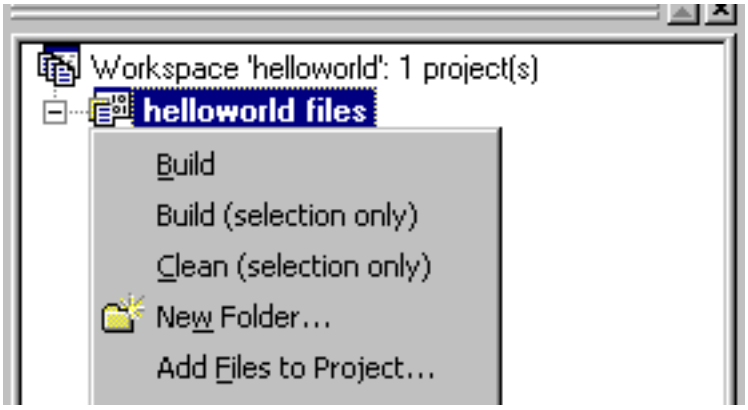

Figure 4

Change the file type to "All files" and select helloworld.asm and helloworld.mak (see figure 5).

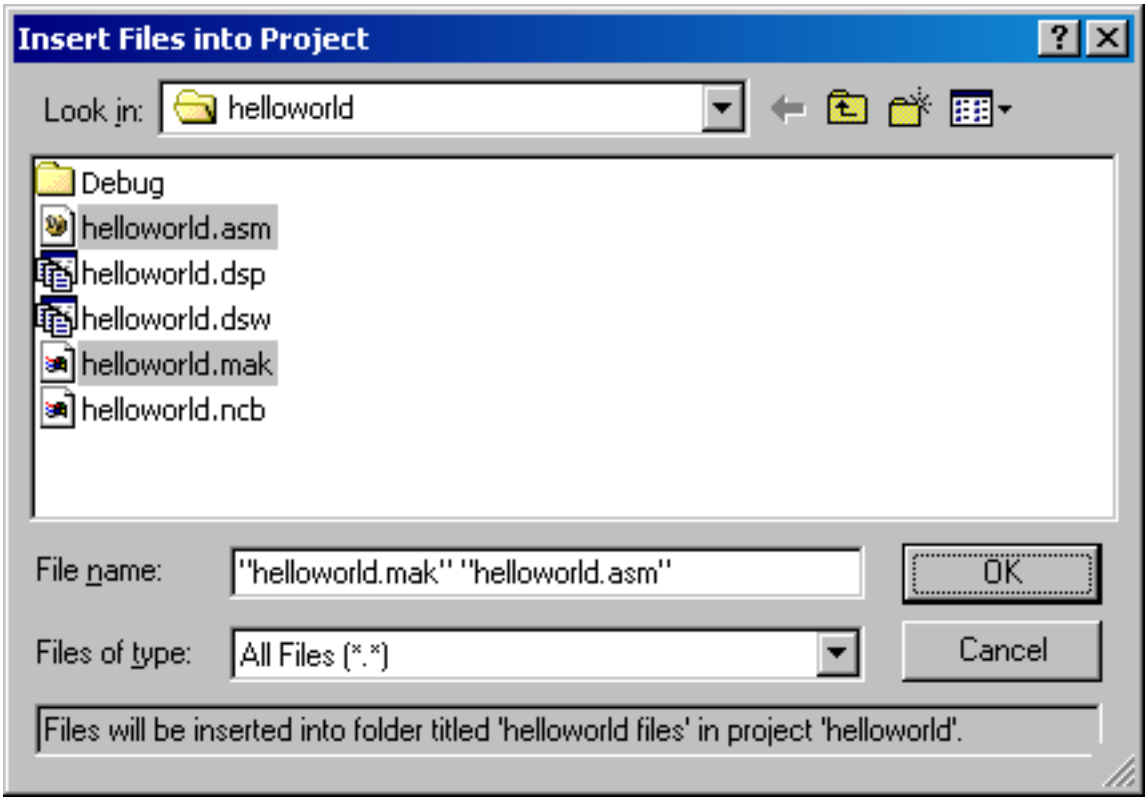

Figure 5

Now your workspace should look something like figure 6.

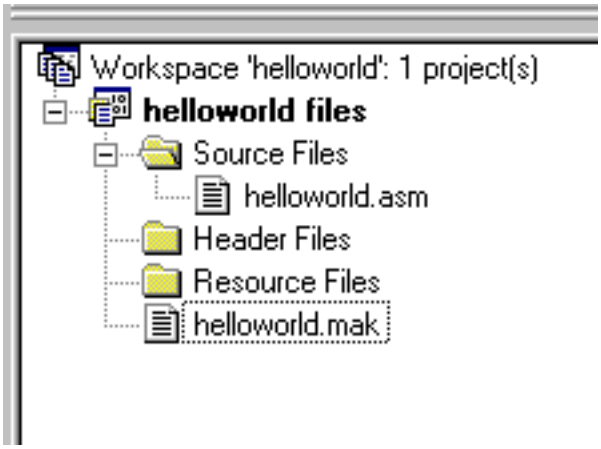

Figure 6

Our final step before we compile is to add paths for executable, library, and include files. This is accomplished via (TOOLS->OPTIONS- >DIRECTORIES). First add the C:\MASM615 path to the executable list (figure 7). Then add C:\MASM615\INCLUDE to the include path (figure 8). Finally add C:\MASM615\LIB to the library path (figure 9). We won't use the include path for this program, but still set it up while we are there.

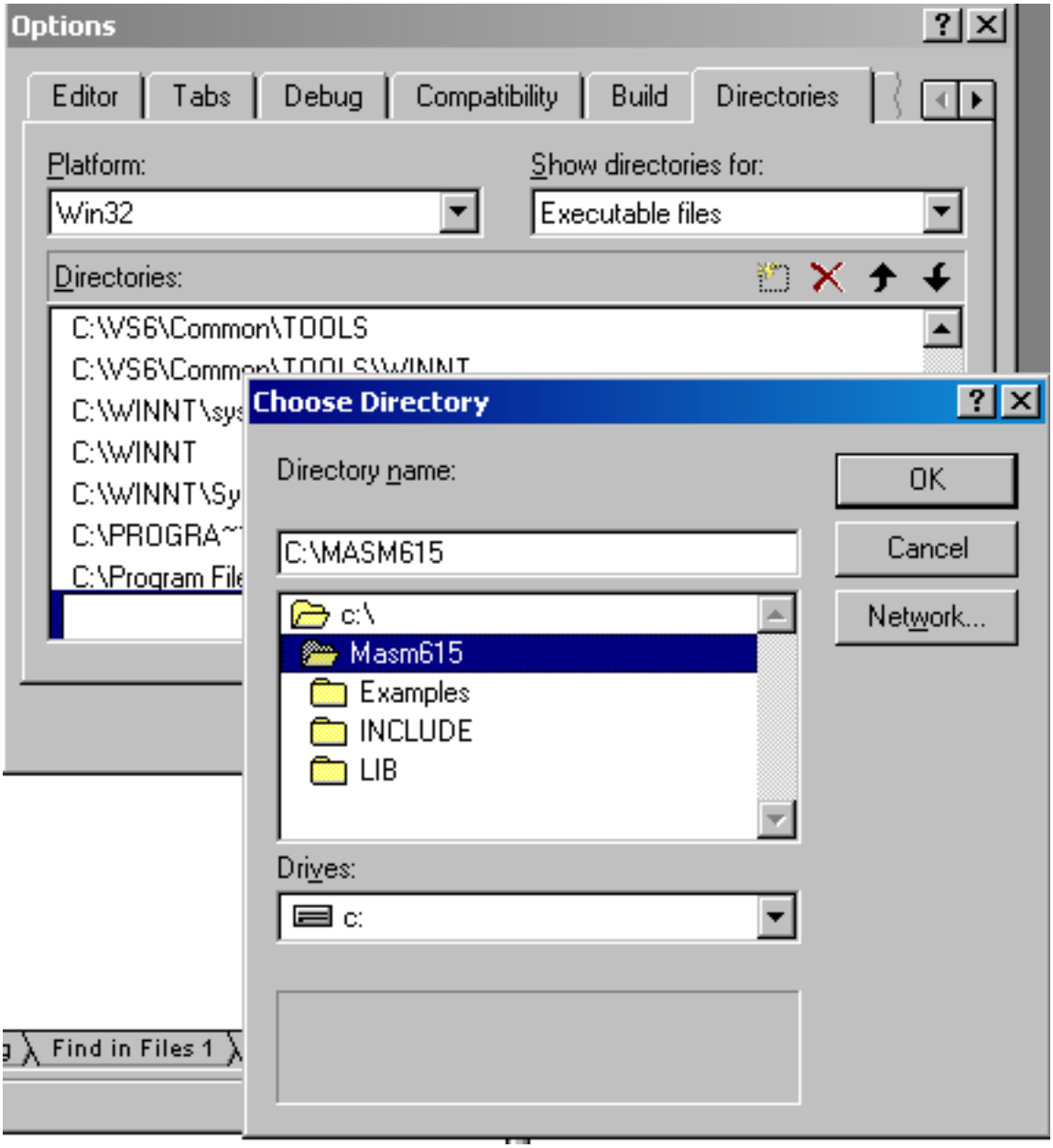
Options
Editor
Tabs
Debug
Compatibility
Build
Directories
Platform:
Win32
Show directories for:
Executable files
Directories:
C:\VS6\Common\TOOLS
C:\WINNT
C:\WINNT\Sy
C:\PROGRA~
C:\Program File
Find in Files 1
Choose Directory
Directory name:
C:\MASM615
OK
Cancel
Network...
c:\
Masm615
Examples
INCLUDE
LIB
Drives:
c:

Figure 7

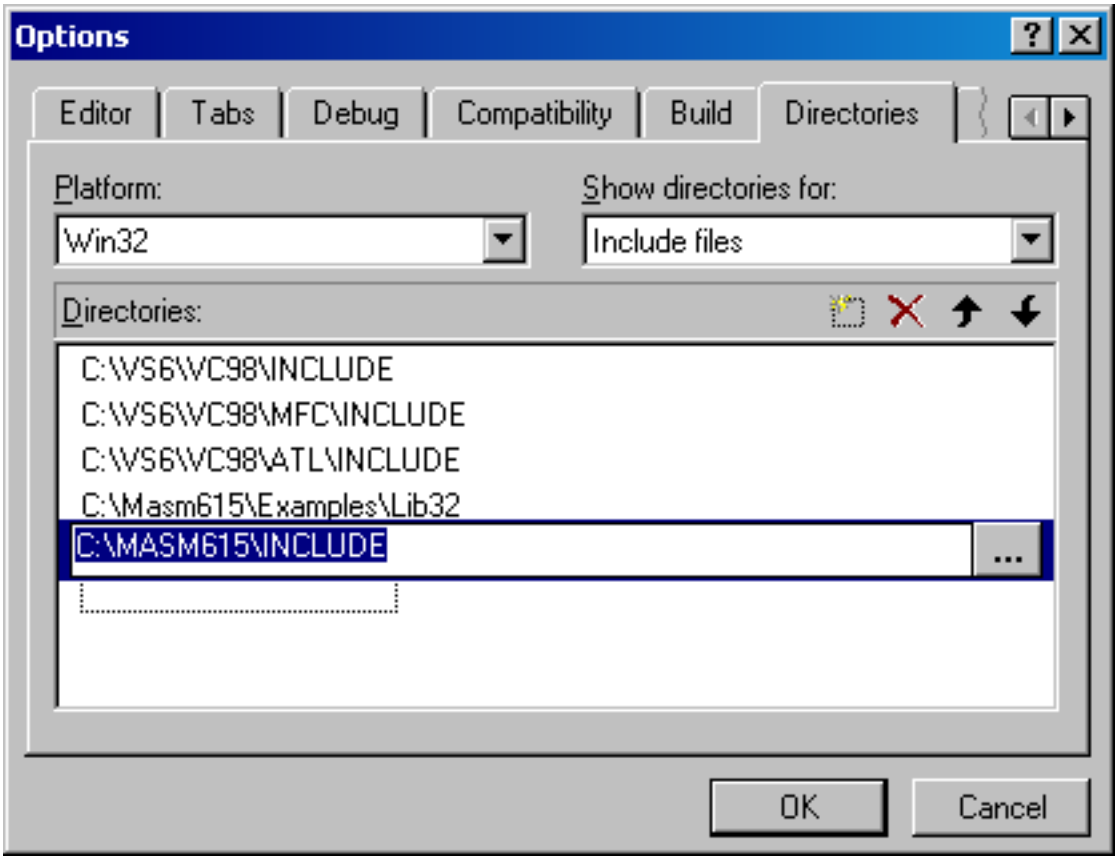
Options
Editor
Tabs
Debug
Compatibility
Build
Directories
Platform:
Win32
Show directories for:
Include files
Directories:
C:\VS6\VC98\INCLUDE
C:\VS6\VC98\MFC\INCLUDE
C:\VS6\VC98\ATL\INCLUDE
C:\Masm615\Examples\Lib32
C:\MASM615\INCLUDE
...
OK
Cancel

Figure 8

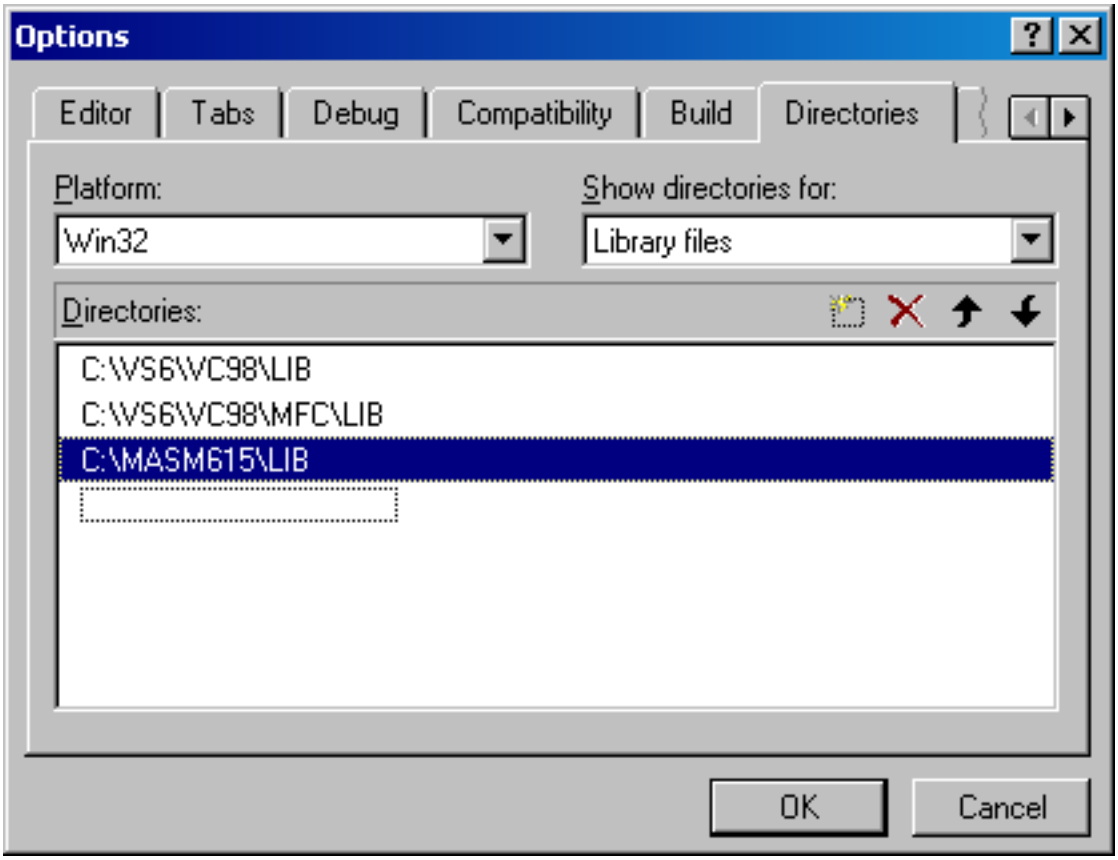

Figure 9

Great! You've successfully established a framework in which to compile and debug your assembly programs. Now we'll continue to the next section to discuss compilation.

Compiling your project

This section takes a takes a brief moment to describe the assembly program and makefile that comprise the package. We'll compile and run our program at the end.

Let's take another look at that assembly program and dissect it a bit. I'll leave the real discussion of assembly to finer folks than I and your text books. But I will give a quick rundown.

```
 1 ; This is a very simple 32 bit assembly program
 2 ; that uses the Win32 API to display hello world on the
 3 ; console.
 4
 5 TITLE Hello World in Win32 ASM       (helloworld.asm)
 6
 7 .386
 8 .MODEL flat, stdcall
 9 .STACK 4096
10
11 ; ---------------------------------------------------------------------------
12 ; These are prototypes for functions that we use
13 ; from the Microsoft library Kernel32.lib.
14 ; ---------------------------------------------------------------------------
```

```
15
16 ; Win32 Console handle
17 STD_OUTPUT_HANDLE EQU -11                ; predefined Win API constant (magic)
18
19 GetStdHandle PROTO,                  ; get standard handle
20         nStdHandle:DWORD                              ; type of console handle
21
22 WriteConsole EQU <WriteConsoleA>    ; alias
23
24 WriteConsole PROTO,                       ; write a buffer to the console
25         handle:DWORD,                            ; output handle
26         lpBuffer:PTR BYTE,                     ; pointer to buffer
27         nNumberOfBytesToWrite:DWORD,             ; size of buffer
28         lpNumberOfBytesWritten:PTR DWORD,  ; num bytes written
29         lpReserved:DWORD                       ; (not used)
30
31 ExitProcess PROTO,                  ; exit program
32         dwExitCode:DWORD                       ; return code
33
34 ; -----------------------------------------------------------------------------
35
36
37
38
39 ; -----------------------------------------------------------------------------
40 ; global data
41 ; -----------------------------------------------------------------------------
42
43 .data
44 consoleOutHandle dd ?                  ; DWORD: handle to standard output device
45 bytesWritten     dd ?                  ; DWORD: number of bytes written
46 message db "Hello World",13,10,0  ; BYTE: string, with \r, \n, \0 at the end
47
48 ; -----------------------------------------------------------------------------
49
50
51
52
53 .code
54
55 ; -----------------------------------------------------------------------------
56 procStrLength PROC USES edi,
57         ptrString:PTR BYTE ; pointer to string
58 ;
59 ; walk the null terminated string at ptrString
60 ; incrementing eax. The value in eax is the string length
61 ;
62 ; parameters: ptrString - a string pointer
63 ; returns: EAX = length of string prtString
64 ; -----------------------------------------------------------------------------
65         mov edi,ptrString
66         mov eax,0                             ; character count
67 L1:                              ; loop
68         cmp byte ptr [edi],0      ; found the null end of string?
69         je  L2                      ; yes: jump to L2 and return
70         inc edi                      ; no: increment to next byte
71         inc eax                      ; increment counter
72         jmp L1                       ; next iteration of loop
73 L2: ret                          ; jump here to return
74 procStrLength ENDP
75 ; -----------------------------------------------------------------------------
76
77
78
79
80 ; -----------------------------------------------------------------------------
81 procWriteString proc
82 ;
83 ; Writes a null-terminated string pointed to by EDX to standard
84 ; output using windows calls.
```

```
85 ; --------------------------------------------------------------------------
86         pushad
87
88         INVOKE procStrLength,edx              ; return length of string in EAX
89         cld                              ; clear the direction flag
90                                          ; must do this before WriteConsole
91
92         INVOKE WriteConsole,
93             consoleOutHandle,               ; console output handle
94             edx,                     ; points to string
95             eax,                     ; string length
96             offset bytesWritten,            ; returns number of bytes written
97             0
98
99         popad
100         ret
101 procWriteString endp
102 ; --------------------------------------------------------------------------
103
104
105
106
107 ; --------------------------------------------------------------------------
108 main PROC
109 ;
110 ; Main procedure. Just initializes stdout, dumps the string, and exits.
111 ; --------------------------------------------------------------------------
112         INVOKE GetStdHandle, STD_OUTPUT_HANDLE ; use Win32 to put
113                                                 ; stdout handle in EAX
114
115         mov [consoleOutHandle],eax             ; Put the address of the handle in
116                                                 ; our variable
117
118         mov edx,offset message                 ; load the address of the message
119                                                 ; into edx for procWriteString
120
121         INVOKE procWriteString                 ; invoke our write string method.
122                                                 ; It'll check EDX
123
124         INVOKE ExitProcess,0                   ; Windows method to quit
125
126 main ENDP
127 ; --------------------------------------------------------------------------
128
129 END main
```

break down by line

- 1-3 are comments. Note that comments begin with a ';' character
- 5-32 are header data including directives, function prototypes, and constants.
- 44-46 are global variables
- 56-101 are procedures to calculate a string length and write a string to the console
- 108-126 are the main procedure that starts the program. We'll break it down further
    - 112, 115 uses the windows api to get an address to write to standard out. Then we store it in a global variable.
    - 118, 121 calls our procWriteString procedure to display the global string "Hello World".

    - 124 calls Windows' ExitProcess to quit the program gracefully.

- 80 indicates the end of the program and file.

Now that we can see the basic form of an assembly program it should be easy to reference it's constructs in manuals and build on the at basic shell. Now, let's have a second look at the helloworld.mak makefile.

```
 1 # A very simple make file for a windows 32 bit assembly console program
 2 # it assembles and links
 3
 4 # nmake help is online at:
 5 #  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcug98/html/_asug_macros_and_
 6
 7 # Assemble the code into coff format producing map and listing files,
 8 # including symbolic debugging info. Try "ml /?" for more options
 9 # and descriptions
10
11 # 32 bit link our .obj file with the kernel32.lib file and create an exe file
12
13 all: helloworld.exe
14
15 helloworld.exe: helloworld.asm
16         ml /nologo /coff /c /Zi /Fl /Fm $?
17         link32 /nologo /DEBUG /incremental:no /subsystem:console /entry:main /out:debug\helloworl
```

If you've ever worked with source distributions this format should look vaguely familiar. But let's detail it.

- 1-1 are comments. Comments begin with the '#' character Comments are goooodddddd!

- 13 defines the default target 'all' and says that it depends on a sub-target helloworld.exe.

- 15 defines the target 'helloworld.exe' and specifies that it depends on the helloworld.asm file. That means nmake will be smart enough to know to recompile if you change the ASM file.

- 16 is the first command run for the 'helloworld.exe' target. It runs the assembler on helloworld.asm to create a coff object file.

- 17 is the second command run for the 'helloworld.exe' target. It runs the 32bit linker to link helloworld.obj with kernel32.lib to make a real program, helloworld.exe!!

Now that we have a better idea how the assembly file is formed and how the makefile helps compile it let's move on the glory moment, compilation and execution. If everything is setup correctly you should be able to click Build icon on the toolbar to compile the program (see figure 10).

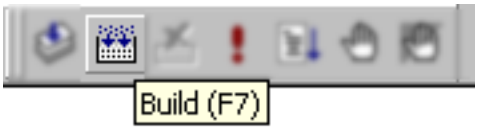

Figure 10

The build panel should look like figure 11.

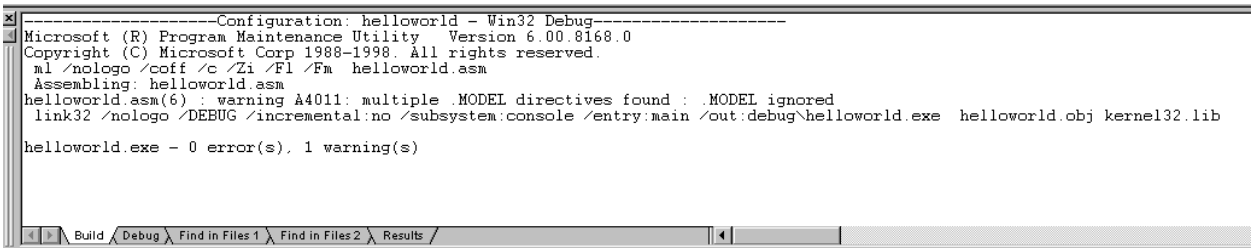

Figure 11

Finally, the climax. Run your compiled program by clicking the execute button (!) (see figure 12).

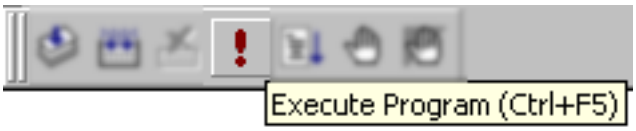

Figure 12

If all goes well you'll see a DOS/CMD box like in the following picture. (see figure 13).

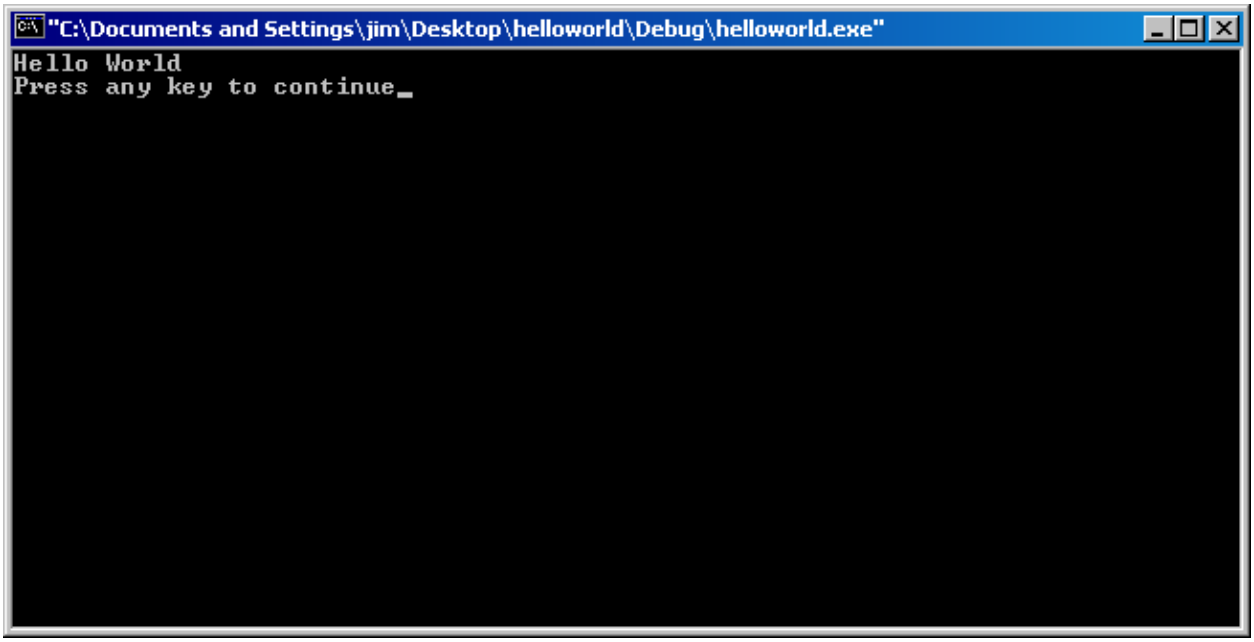

Figure 13

Excellent! Now we know our framework is capable of compiling and running an assembly program and we have a working example of code and a make file. We're ready to move on to debugging the program and examining its guts.

Debugging your project

Debugging can be an especially powerful tool for learning programming. It is essentially the process of walking through your program in different orders and examining the contents of the computer's storage areas. We'll take a brief walk through a typical debug session using helloworld.exe.

We start our debugging by setting a break point. This is a line of code where the execution should pause until you

decide to continue. From this point to can examine the contents of CPU registers or memory. Put your cursor on line 112 and click the Add Breakpoint tool button. Notice the red dot that appears next to the line to signal the breakpoint. (see figure 14)

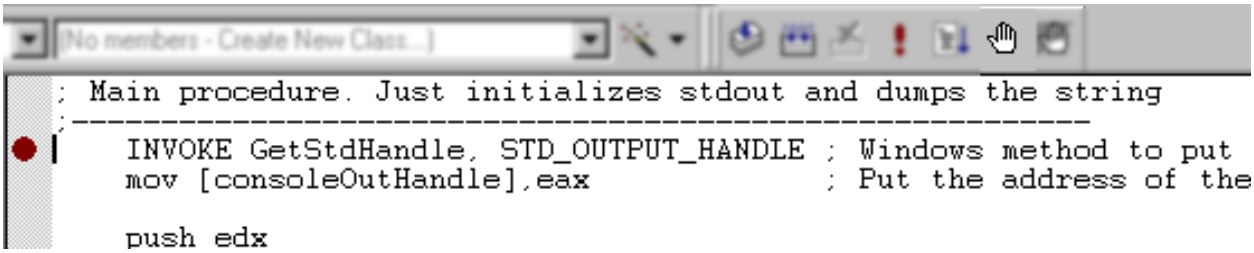

Figure 14

Now start the debugger by clicking the debug toolbar icon (see gigure 15)

Figure 15

Let's take a moment to notice a couple of things. Once you start the debugger. A couple of new debug windows will appear (depending on your configuration) (see figure 16). This is a good view for C++, but isn't great for assembly. Right click a frame of one of the new windows and make it such that you have the "Registers" and "Memory" docklets visible (see figure 17).

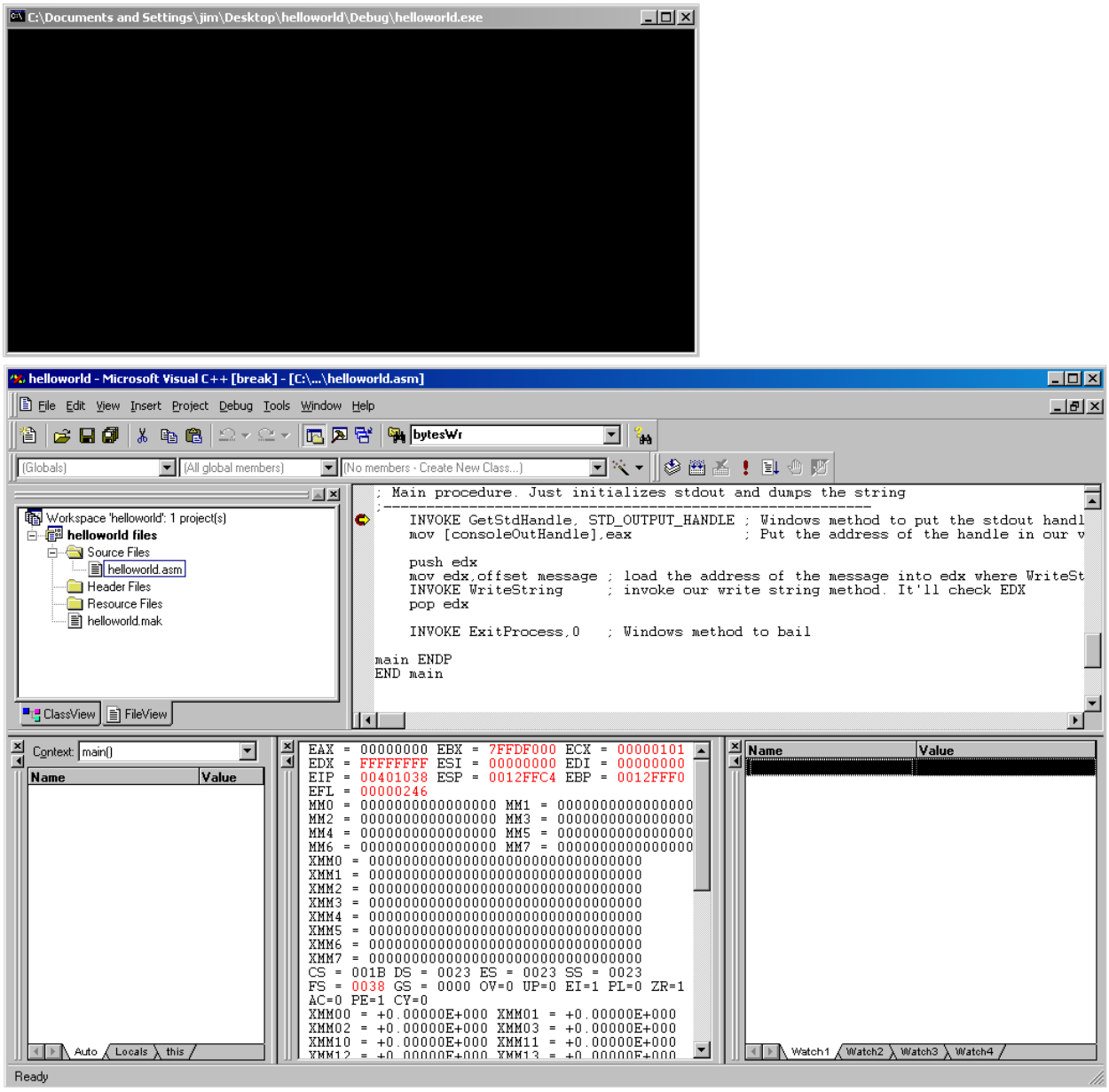
C:\Documents and Settings\jim\Desktop\helloworld\Debug\helloworld.exe
helloworld - Microsoft Visual C++ [break] - [C:\...\helloworld.asm]
File Edit View Insert Project Debug Tools Window Help
bytesWr
(Globals)
(All global members)
(No members - Create New Class...)
Workspace 'helloworld': 1 project(s)
helloworld files
Source Files
helloworld.asm
Header Files
Resource Files
helloworld.mak
ClassView
FileView
; Main procedure. Just initializes stdout and dumps the string
INVOKE GetStdHandle, STD_OUTPUT_HANDLE ; Windows method to put the stdout handl
mov [consoleOutHandle],eax ; Put the address of the handle in our v
push edx
mov edx,offset message ; load the address of the message into edx where WriteSt
INVOKE WriteString ; invoke our write string method. It'll check EDX
pop edx
INVOKE ExitProcess,0 ; Windows method to bail
main ENDP
END main
Context: main()
Name
Value
Auto
Locals
this
EAX = 00000000 EBX = 7FFDF000 ECX = 00000101
EDX = FFFFFFFF ESI = 00000000 EDI = 00000000
EIP = 00401038 ESP = 0012FFC4 EBP = 0012FFF0
EFL = 00000246
MM0 = 0000000000000000 MM1 = 0000000000000000
MM2 = 0000000000000000 MM3 = 0000000000000000
MM4 = 0000000000000000 MM5 = 0000000000000000
MM6 = 0000000000000000 MM7 = 0000000000000000
XMM0 = 00000000000000000000000000000000
XMM1 = 00000000000000000000000000000000
XMM2 = 00000000000000000000000000000000
XMM3 = 00000000000000000000000000000000
XMM4 = 00000000000000000000000000000000
XMM5 = 00000000000000000000000000000000
XMM6 = 00000000000000000000000000000000
XMM7 = 00000000000000000000000000000000
CS = 001B DS = 0023 ES = 0023 SS = 0023
FS = 0038 GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=1
AC=0 PE=1 CY=0
XMM00 = +0.00000E+000 XMM01 = +0.00000E+000
XMM02 = +0.00000E+000 XMM03 = +0.00000E+000
XMM10 = +0.00000E+000 XMM11 = +0.00000E+000
Name
Value
Watch1
Watch2
Watch3
Watch4
Ready

Figure 16

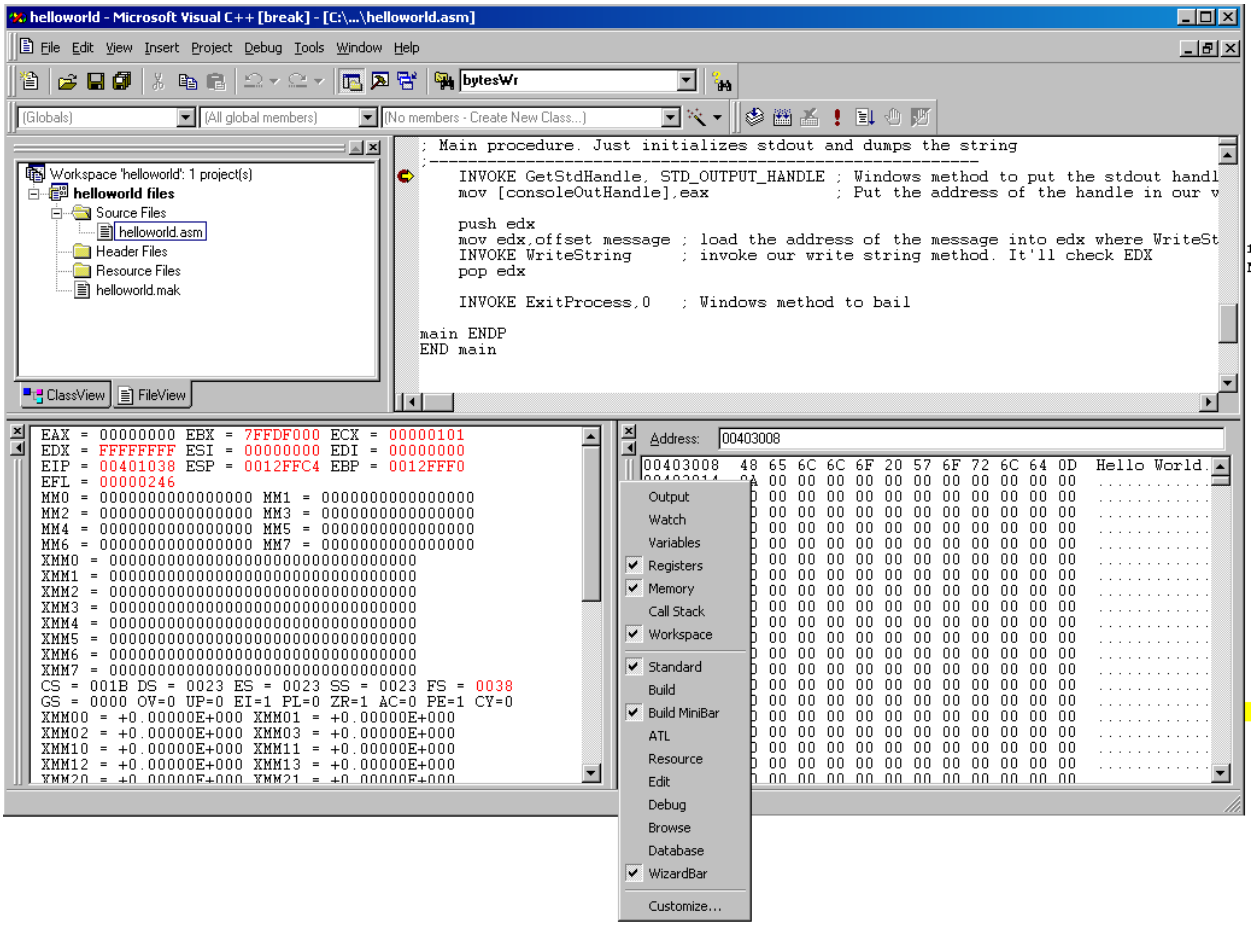

Figure 17

You can use the "Step Over" button on the debug toolbar to execute the next line (see figure 18). Watch the registers panel when you do that. Continue to click that button until you get to line 121 where "procWriteString" is invoked. Look at the value of the EDX register in the registers window. Cut and paste it's value into the address box of the memory window. You'll see our message "Hello World" is stored at that memory address. This tells you that the value in EDX is the address of the our string variable (figure 19). Pretty handy huh?

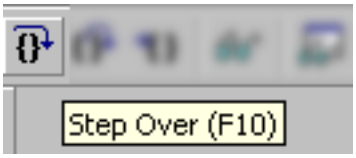

Figure 18

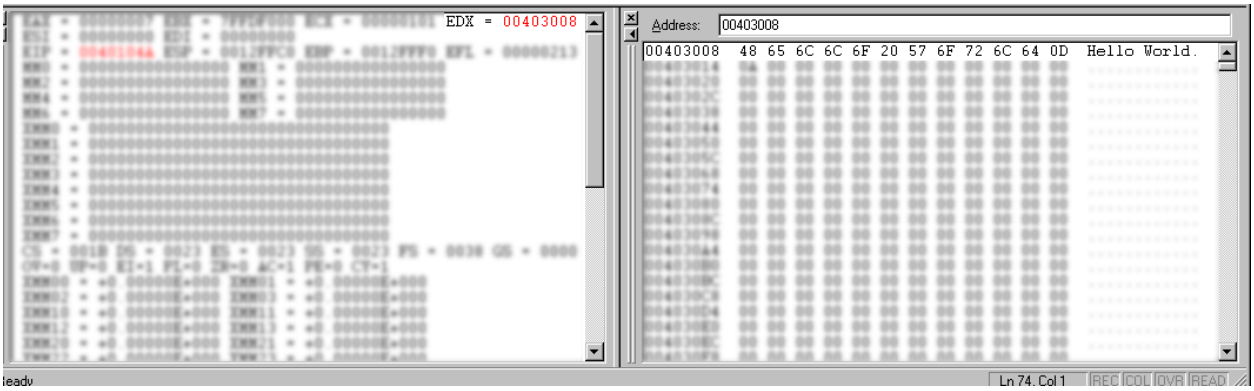

Figure 19

Now, that you've seen the meat of debugging you can halt the debugger, let the program run to completion, or continues stepping to the end of the code. You should now understand how to use debugging to examine the contents of a running program.

Conclusion

We've now gone completely from idea, to code, to testing using MS Visual C++ and MASM for development. This frame work of assembly file and make file is enough for us to spring board into other more interesting projects. There is still a lot to learn about processors and different architectures. This quick start should provide you with the stepping stones you need to continue studying the ms/intel architecture.

References

These are books, papers, and sites that I drew on for this article.

List of References.

1. "Assembly Language for Intel Based Computers" 4th ed by Kip R. Irvine [http://www.nuvisionmiami.com/books/asm/index.html]

2. MS Assembler Ref [http://msdn.microsoft.com/library/en-us/vcmasm/html/vcoriMicrosoftAssemblerMacroLanguage.asp]

3. MS nmake ref [http://msdn.microsoft.com/library/en-us/vcug98/html/_asug_overview.3a_.nmake_reference.asp]

4. Intel Pentium 4 Architecture Ref [http://www.intel.com/design/pentium4/MANUALS/index.htm]