LAB 3

THE ASSEMBLE-LINK-EXECUTE CYCLE USING COMMAND LINE TOOLS

Directions. Answer the questions marked ⍰. Turn in this paper when you finish.

INTRODUCTION

So far, you have been developing assembly language programs in Visual Studio. Visual Studio is an example of an *integrated development environment (IDE)*, a software engineering tool that combines a code editor, debugger, build system, and other development tools into a streamlined, graphical user interface.

You do *not* have to use Visual Studio to develop assembly language programs. The Microsoft Macro Assembler and Microsoft Incremental Linker are actually command-line tools. Visual Studio is using them behind the scenes—you see their output every time you assemble a program in Visual Studio—but you can also use them directly from the Windows Command Prompt, without starting Visual Studio.

You can run the Microsoft Macro Assembler and Microsoft Incremental Linker from an ordinary Windows command prompt (Start > Accessories > Command Prompt), but they are located in a nonstandard directory, so it takes some extra work to do this. Fortunately, when Visual Studio was installed, it added a "Visual Studio Command Prompt" item to the Start menu, which starts a command prompt and does configures it for you, so you don't have to do it manually. [1]

In this lab, you will learn how to write, assemble, and link programs *without* using Visual Studio. Instead, you'll run the assembler and linker from the command prompt, and you'll use Windows Notepad to write your assembly language code.

This will also help to clarify the assemble-link-execute cycle, which was discussed in the assigned reading from Section 3.3, "Assembling, Linking, and Running Programs."

---

[1] If you wanted to use an ordinary Windows Command Prompt and configure it manually, for this lab, you would need to change the PATH environment variable to include the Visual C++ *bin* directory, e.g., C:\Program Files\Microsoft Visual Studio 10.0\VC\bin. The "Visual Studio Command Prompt" item in the Start menu starts a Command Prompt and runs a batch file that, among other things, locates this directory and sets the PATH automatically.

1. GETTING ACQUAINTED WITH THE COMMAND LINE TOOLS

☐ **Open the Visual Studio Command Prompt.** Click Start > All Programs > CSSE Apps > Microsoft Visual Studio 2010 > Visual Studio Tools > Visual Studio Command Prompt. A window will open with a command prompt like the following.

```
C:\Program Files\Microsoft Visual Studio 10.0\VC>
```

☐ **Maximize the window.** Press Alt+Spacebar to open the system menu, then press X to select Maximize from the menu. The window should occupy the full height of the screen.

☐ **Change to your local Documents directory (on the hard drive, not a network drive).** At the command prompt, type this, replacing *yourID* with your Auburn ID:

```
cd C:\Users\yourID\Documents
```

The new directory should be shown in the command prompt.

```
C:\Users\yourID\Documents>
```

☐ **Create a new directory for your files, and change to that directory.** Type:

```
mkdir Lab3
cd Lab3
```

The new directory should be shown in the command prompt.

```
C:\Users\yourID\Documents\Lab3>
```

☐ **Create a file named *hello.asm*.** At the command prompt, type:

```
notepad hello.asm
```

Notepad will open and ask, "Cannot find the hello.asm file. Do you want to create a new file?" Click Yes.

(Windows Notepad opens and saves plain text files. You could also use another text editor, like Notepad++, Sublime Text, or Vim. The point is that *.asm* files are just ASCII text files; you can write them in any text editor, not just Visual Studio.)

☐ **In Notepad, type the following program. Then, save it, and exit Notepad.**

```
INCLUDE Irvine32.inc

.data
hello BYTE "Hello", 0

.code
main PROC
    mov edx, OFFSET hello
    call WriteString
    exit
main ENDP
END main
```

☐ **Get a directory listing.** Back in the Visual Studio Command Prompt, type:

```
dir
```

Make sure *hello.asm* is listed.

2. ASSEMBLING

At this point, you should have a Visual Studio Command Prompt window open. The prompt should be `C:\Users\`*yourID*`\Documents\Lab3>`*, and when you type "dir", your* hello.asm *file should be included in the directory listing.*

☐ **Get help for the *ml* command.** At the command prompt, type:

```
ml /?
```

1. What program is ml.exe? _____
2. What does the /c option do? _____
3. What does the /I option do? _____

☐ **Get a directory listing** by typing `dir`, as before. (You will need to refer to it later.)

☐ **Assemble your program.** At the command prompt, type:

```
ml /c /IC:\Irvine hello.asm
```

☐ **Get a directory listing** again. Compare it to the listing before you ran *ml.exe*. What new file did *ml.exe* just create?

3. LINKING

At this point, you should have a Visual Studio Command Prompt window open. The prompt should be `C:\Users\`*yourID*`\Documents\Lab3>`*, and when you type "dir", the listing should include a file named* hello.asm *and another named* hello.obj.

☐ **Try to link your .obj file into an executable.** At the command prompt, type:

```
link hello.obj
```

1. What program is link.exe? _____
2. What error do you get? _____

☐ **Try to link your .obj file again.** At the command prompt, type:

```
link /SUBSYSTEM:CONSOLE hello.obj
```

What errors do you get (summarize them in just a few words)?

Recall that Windows (like every operating system) provides an *application programming interface*—an *API.* This API provides functions that allow application programs to read and write files, read and write to the display, allocate memory, etc.

ExitProcess is a Windows API function that is used to terminate a program. The machine code for this function is located in C:\Windows\system32\kernel32.dll. In your assembly language program, the `exit` line is actually a *macro* (essentially, a shorthand notation) that the assembler expands into a *ExitProcess* call.

WriteString is part of the library provided by your textbook's author (Kip Irvine). The machine code for this function is located in C:\Irvine\irvine32.lib.

☐ **Try to link your .obj file again.** At the command prompt, type this (all on one line):

```
link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine irvine32.lib
   kernel32.lib hello.obj
```

(The **bold** text is different from the last `link` command you typed.)

What error do you get (summarize it in just a few words)?

MessageBoxA is a Windows API function that is used to display a message dialog box (with an OK button). The machine code for this function is located in C:\Windows\system32\user32.dll. Although you are not using this function, it is used by another function in Kip Irvine's library (*MsgBox*). Kip Irvine's library is a *statically linked library.* All of the functions in his library will be linked into your executable—even ones you don't actually use, like *MsgBox*.

☐ **Try to link your .obj file again.** At the command prompt, type this (all on one line):

```
link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine irvine32.lib
   kernel32.lib user32.lib hello.obj
```

☐ **Get a directory listing.** Compare it to the listing from earlier.

What new file did *link.exe* just create? _____

☐ **Run your program.** At the command prompt, type:

```
hello
```

Or, you could also type: `hello.exe`

☐ **Use *dumpbin* to see what dynamically linked libraries (DLLs) are required for your executable (.exe file) to run, and what functions it references in those DLLs.** Type:

```
dumpbin /IMPORTS hello.exe
```

What DLLs are required for *hello.exe* to run?

Kip Irvine's *WriteString* function calls a Windows API function named *WriteConsoleA*. In what DLL is the *WriteConsoleA* function located?

4. MAKING CHANGES

At this point, you should have a Visual Studio Command Prompt window open. The prompt should be `C:\Users\`*yourID*`\Documents\Lab3>`*, and when you type "dir", the directory listing should include files named* hello.asm*,* hello.obj, *and* hello.exe.

☐ **Open *hello.asm* again in Notepad.** At the command prompt, type:

```
notepad hello.asm
```

☐ **Modify *hello.asm*, save it, and exit Notepad.** Change the BYTE declaration to:

```
hello BYTE "This has changed", 0
```

☐ **Get a directory listing,** and make sure the modification time for *hello.asm* is correct (it should be the current time, since you just saved it a few seconds ago).

☐ **Run your program. (Do not run *ml* or *link* yet.)** At the command prompt, type:

```
hello
```

The output should be "Hello," not "This has changed."

Why don't you get the output "This has changed"?

☐ **Assemble your program.** At the command prompt, type:

```
ml /c /IC:\Irvine hello.asm
```

☐ **Get a directory listing,** and note the modification time of *hello.obj*. The assembler just assembled your *hello.asm* file and overwrote *hello.obj.*

☐ **Run your program.** At the command prompt, type:

```
hello
```

The output should still be "Hello," not "This has changed."

Why don't you get the output "This has changed"?

☐ **Link your program.** At the command prompt, type (all on one line):

```
link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine irvine32.lib
      kernel32.lib user32.lib hello.obj
```

☐ **Get a directory listing,** and make sure the modification time for *hello.exe* is correct (it should be the current time, since *link.exe* just recreated it).

☐ **Run your program.** At the command prompt, type:

```
hello
```

The output should be "This has changed."

5. USING NMAKE

At this point, you should have a Visual Studio Command Prompt window open. The prompt should be `C:\Users\`*yourID*`\Documents\Lab3>`*, and when you type "dir", you should have files named* hello.asm*,* hello.obj, *and* hello.exe.

Many developers prefer to work with command line tools, rather than using an IDE like Visual Studio. But typing the *ml* and *link* commands can be incredibly tedious (and error-prone). So, developers typically use a *build tool,* which automates this process. In this section, you will learn how to use NMAKE, the command-line build tool that is included with Visual Studio.

☐ **Open Notepad to begin editing a new file.** At the command prompt, type:

```
notepad
```

☐ **Type the following in Notepad.** The right arrow → indicates a tab. You **must** use tab characters where indicated, not spaces (so don't copy and paste this code from the PDF).

```
all: hello.exe

hello.exe: hello.asm
  →  ml /c /IC:\Irvine hello.asm
  →  link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine \
  →    →  irvine32.lib kernel32.lib user32.lib hello.obj

clean:
  →  del /S hello.exe hello.obj
```

☐ **Click File > Save As.** For the filename, type this, **with** the double-quotes:

"C:\Users*yourID*\Documents\Lab3\makefile"

Click OK to save the file. Then, exit Notepad.

☐ **Get a directory listing.** Back in the Visual Studio Command Prompt, type:

```
dir
```

Make sure the file you just created is listed and named *makefile* (not *makefile.txt*).

☐ **Clean your project using *nmake*.** Back in the Visual Studio Command Prompt, type:

```
nmake clean
```

Note that it runs the *del* command you typed in the makefile (above), which deletes your executable (hello.exe) and object file (hello.obj).

☐ **Build your project using *nmake*.** Type:

```
nmake
```

You could also have typed `nmake all`

Notice what happened: It ran the *ml* and *link* commands that you typed in the makefile.

☐ **Immediately try to build your project using *nmake* again.** Type:

```
nmake
```

Notice that it did **not** run the assembler or the linker this time!

Briefly, here's what's happening:

- The makefile contains three **targets**: *all*, *hello.exe*, and *clean* (shown in bold below). When you typed "nmake clean", you were instructing NMAKE to build the *clean* target. When you typed "nmake" without a specific target, it built the *all* target by default.

```
all: hello.exe

hello.exe: hello.asm
 → ml /c /IC:\Irvine hello.asm
 → link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine \
 → → irvine32.lib kernel32.lib user32.lib hello.obj

clean:
 → del /S hello.exe hello.obj
```

- The colons indicate **dependencies** (shown below). To build *all*, you need to build *hello.exe.* To build *hello.exe*, you need *hello.asm*.

```
all: hello.exe

hello.exe: hello.asm
...
```

- The tab-indented lines under a target list the commands that need to be run to build that target. Since the *link* command is excessively long, we used a backslash \ to split it into two lines.

```
hello.exe: hello.asm
 → ml /c /IC:\Irvine hello.asm
 → link /SUBSYSTEM:CONSOLE /LIBPATH:C:\Irvine \
 → → irvine32.lib kernel32.lib user32.lib hello.obj
```

- NMAKE looks at the modification times of files to determine what to build. (These are the times that were displayed by the *dir* command above.) As an example:
  - As specified in the makefile, *hello.exe* depends on *hello.asm.*
  - If *hello.asm* has been modified more recently than *hello.exe*, or if *hello.exe* does not exist, then nmake will run the *ml* and *link* commands try to create *hello.exe*.
  - If *hello.exe* has been modified more recently than *hello.asm*, then it is up-to-date, and nmake will not run the *ml* and *link* commands.
- Since *ml* is listed first, followed by *link*, the two commands will be run in that order. If *ml* raises an error, nmake will stop without running *link.* The build is only considered "successful" if both *ml* and *link* exit without producing an error.

Now, let's get some more experience with *nmake.*

☐ **Modify *hello.asm*, save it, and exit Notepad.** Change the BYTE declaration to:

```
hello BYTE "I changed this again", 0
```

☐ ⓘ **Get a directory listing.** What are the modification times for:

hello.asm _____

hello.exe _____

☐ ⓘ **Run nmake.** Does it run *ml* and *link*? Why?

☐ ⓘ **Get a directory listing.** What are the modification times for:

hello.asm _____

hello.exe _____

☐ ⓘ **Run nmake again.** Does it run *ml* and *link*? Why?

☐ **Introduce an error into *hello.asm*, save it, and exit Notepad.** Change the declaration:

```
hello BYYYYYYYYYTE "I changed this again", 0
```

☐ ⓘ **Run nmake.**

Does it run *ml*? _____

Does it run *link*? _____

☐ **Fix the error in *hello.asm*.**

☐ ⓘ **Run nmake.**

Does it run *ml*? _____

Does it run *link*? _____

CONCLUSION

Realistically, you will probably want to use Visual Studio for the rest of the semester. Its debugger is especially useful. But now you know what it's doing under the hood: it's just calling *ml* and *link*.

Now, you also know that you can develop assembly language programs with just *ml*, *link*, and a text editor... and perhaps *nmake*, for when you get tired of typing that *link* line...