APPENDIX B

Environment Setup for Assembly Programming

Objective

Appendix B describes how to prepare your Development Environment for programming in Assembly. Examples are provided for Windows using Visual Studio and MASM, for macOS using Xcode (Clang/LLVM, GAS compatible), and for Linux using NASM and ld. We offer some basic setup configurations and you can explore other options if desired. The code examples we use are 32-bit.

Windows – Visual Studio (2017) – MASM

1. Install Visual Studio if needed.
2. Open Visual Studio and create a new empty C++ project, giving the project an appropriate name.
3. Once the project has been created and is open, right-click on the project in the Solution Explorer and select "Build Dependencies" → "Build Customizations"
4. Check the "masm" box.
5. In the Solution Explorer, right-click "Source Files" then select "Add" → "New item…"
6. Under "Visual C++" click "Utility" then choose "Text File." When you name the file also type the *.asm* extension.
7. In the Solution Explorer, right-click on the newly created *.asm* file, select "Properties," and under "General" change "Item Type" to "Microsoft Macro Assembler" if not already set.
8. In the Solution Explorer, right-click on the project, select "Properties" and within the Properties you should see a drop-down menu called "Microsoft Macro Assembler." If the menu is missing, return to Step 2. If the menu exists, continue to Step 9.
9. In the project Properties drop-down menu, navigate to "Linker" → "System" and in the "SubSystem" drop-down box select "Windows (/SUBSYSTEM:WINDOWS)"
10. In the project Properties drop-down menu, navigate to "Linker" → "Advanced" and in the "Entry Point" box type either "main" or "_main" depending on your preference for identifying the main function in your Assembly code. We use "_main" throughout the book.
11. Write a sample Assembly program in the *.asm* file.

```
; First Program for MASM

.386
.MODEL FLAT, stdcall
.STACK 4096

ExitProcess PROTO, dwExitCode:DWORD

.DATA
num DWORD 80
sum DWORD ?

.CODE
_main PROC
   mov eax, num
   add eax, 20
   mov sum, eax

   INVOKE ExitProcess, 0
_main ENDP
END
```

12. Set a breakpoint at a suitable location (e.g., `mov eax, num`).
13. Build and Run the program.
14. When the program halts at the breakpoint, arrange the window frames to your preference. We recommend opening the following windows in the "Debug" → "Windows" menu.
    - Registers
    - Memory (choose at least Memory 1, others are optional)
    - Disassembly
    - Autos
15. Another useful option is to right-click anywhere in the Registers window and select "Flags," which enables viewing of the *flags* register. Select any other registers you have interest in watching.
16. To make Visual Studio create a listing file when assembling your code, in the project Properties navigate to "Microsoft Macro Assembler" → "Listing File" and set the following options.
    - "Generate Preprocessed Source Listing" to "Yes"
    - "List All Available Information" to "Yes"
    - "Assembled Code Listing File" to "$(ProjectName).lst" or any desired file path

    After re-running the program to create a listing file, you may need to set the "Generate Preprocessed Source Listing" back to "No" so debugging works correctly.
17. If you re-run the program you will find the listing file created in:

    ProjectFolder\ProjectName\ProjectName.lst (file type is shown as "MASM Listing"). Example file path: C:\Documents\Visual Studio 2017\Projects\masm_testing\ masm_testing\ masm_testing.lst
18. Optional: In the project Properties drop-down menu, navigate to "Linker" → "Advanced" and in the "Image Has Safe Exception Handlers" drop-down box select "No."

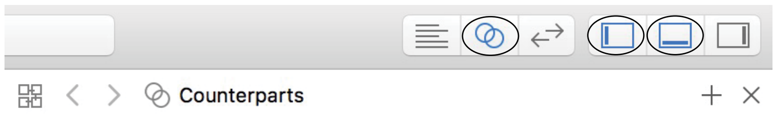
macOS – Xcode (8.2) – Clang/LLVM (GAS compatible)

1. Install Xcode from the AppStore if needed.
2. Open Xcode and create a new project.
3. The template for the new project should be a "macOS" → "Application" and select "Command Line Tool"

4. Give the project an appropriate "Product Name," for example "asm_testing"
5. Language should be set to C++.
6. Create the project in an appropriate folder.
7. In the Project Navigator, select "main.cpp," press the "Delete" key, then select "Move to Trash."
8. In the Project Navigator, right-click on the project folder and select "New File…" then under "macOS" → "Other" (scroll down) chose "Assembly File" (the icon has a large "S"). Click "Next."
9. Name the file, then click "Create."
10. Write a sample Assembly program in the *.s* file.

```
# First Program for GAS, Clang/LLVM

.data
num: .long 80

.bss
.lcomm sum, 4

.text
.globl _main
_main:
   movl num, %eax
   addl $20, %eax
   movl %eax, sum

   pushl $0
   subl $4, %esp
   movl $1, %eax
   int $0x80
.end
```

11. In the Project Navigator, click on the project (top level) and under the "Architectures" drop-down menu and select "32-bit Intel (i386)."
12. Set a breakpoint at a suitable location (e.g., `movl num, %eax`).
13. Build and Run the program.
14. Arrange the windows to your preference, but we recommend the following setup (selections in blue).



15. Registers are viewable in the lower left "Debug Area" and output is in the lower right.
16. To view disassembly, in the "Assistants Area" window (the overlapping circles in the above figure), click on "Counterparts" (see above figure) and select "Disassembly" (at the bottom of the menu).
17. To create a separate listing file, we recommend using `otool`.
    - In Terminal.app, navigate to the folder where the object file for the Xcode project was created (an easy approach is to type "`cd `" in Terminal then navigate to the object file in Finder and drag the folder location into Terminal)
    - Example path:
    - asm_testing/DerivedData/asm_testing/Build/Intermediates/asm_testing.build/Debug/asm_testing.build/Objects-normal/i386/asm_testing.o
    - Run `otool` in Terminal: `otool -dtvj asm_testing.o > disassembly.txt`

- To see flag options to get the output you desire simply type `otool` and press return
- We use `d` = print data section, `t` = print text section, `v` = print verbosely, and `j` = print opcode bytes

Linux – NASM

For the Linux configuration we use Ubuntu. You can adjust the steps for your chosen distribution.

1. Ensure that gcc and g++ are installed by issuing the following commands in the Terminal.
   - `~$ gcc -v`
   - `~$ g++ -v`
2. To ensure you can run both 32-bit and 64-bit programs, you need to install the latest "multilib" files.
   - `~$ sudo apt-get install gcc-5-multilib`
   - `~$ sudo apt-get install g++-5-multilib`
3. Install NASM and verify version (2.11 or higher).
   - `~$ sudo apt-get install nasm`
   - `~$ nasm -v`
4. Use your preferred editor such as *gedit*, *vim*, or *vi* to create a *.s* or *.asm* file in an appropriate folder location and write a sample program.

```
; First Program for NASM

SECTION .data
num: dd 80

SECTION .bss
sum: resd 1

SECTION .text
global _main
_main:
   mov eax, DWORD [num]
   add eax, 20
   mov DWORD [sum], eax

mov eax, 1
mov ebx, 0
int 80h
```

5. Assemble the file using `nasm` and then link using `ld`.
   - `~$ nasm -f elf32 -o testing.o testing.asm`
   - `~$ ld -e _main -melf_i386 -o testing testing.o`
6. Run the program: `~$ ./testing`
7. To debug using the command-line, refer to **Appendix D: Command-Line Debugging Assembly with GDB**.
8. To create listing files and perform other tasks with the object file, you can use utilities such as `objdump` or `otool`, but the simplest way is to use the `-l` flag when assembling.
   - `~$ nasm -f elf32 -o testing.o -l testing.lst testing.asm`