Introduction

The Microsoft® Macro Assembler *Programmer's Guide* provides the information you need to write and debug assembly-language programs with the Microsoft Macro Assembler (MASM), version 6.1. This book documents enhanced features of the language and the programming environment for MASM 6.1.

This *Programmer's Guide* is written for experienced programmers who know assembly language and are familiar with an assembler. The book does not teach the basics of assembly language; it does explain Microsoft-specific features. If you want to learn or review the basics of assembly language, refer to "Books for Further Reading" in this introduction.

This book teaches you how to write efficient code with the new and advanced features of MASM. *Getting Started* explains how to set up MASM 6.1. *Environment and Tools* introduces the integrated development environment called the Programmer's WorkBench (PWB). It also includes a detailed reference to Microsoft tools and utilities such as Microsoft ® CodeView ®, LINK, and NMAKE. The Microsoft Macro Assembler *Reference* provides a full listing of all MASM instructions, directives, statements, and operators, and it serves as a quick reference to utility commands.

For more information on these same topics, see the online Microsoft Advisor, which is a complete reference to Macro Assembler language topics, to the utilities, and to PWB. You should be able to find most of the information you need in the Microsoft Advisor.

New and Extended Features in MASM 6.1

MASM 6.1 continues the break with tradition established by version 6.0. It incorporates conveniences of high-level languages while offering all the traditional advantages of assembly-language programming.

For example, MASM 6.1 includes the Programmer's WorkBench, which provides the same integrated software development environment enjoyed by programmers of Microsoft high-level languages such as C and Basic. From within PWB you can edit, build, debug, or run a program. You can perform most of these operations with either menu selections or keyboard commands. You can also customize PWB to suit your individual programming and editing requirements and preferences.

MASM Features New Since Version 5.1

MASM 6.1 includes several features designed to make programming more efficient and productive. The following list briefly describes how MASM 6.1 improves on the language features of the popular version 5.1.

- MASM 6.1 has many enhancements related to types. You can now use the same type specifiers in initializations as in other contexts (**BYTE** instead of **DB**). You can also define your own types, including pointer types, with the new **TYPEDEF** directive. See Chapter 3, “Using Addresses and Pointers,” and Chapter 4, “Defining and Using Simple Data Types.”
- The syntax for defining and using structures and records has been enhanced since version 5.1. You can also define unions with the new **UNION** directive. See Chapter 5, “Defining and Using Complex Data Types.”
- MASM now generates complete CodeView information for all types. See Chapter 3, “Using Addresses and Pointers,” and Chapter 4, “Defining and Using Simple Data Types.”
- New control-flow directives let you use high-level–language constructs such as loops and if-then-else blocks defined with **.REPEAT** and **.UNTIL** (or **.UNTILCXZ**); **.WHILE** and **.ENDW**; and **.IF**, **.ELSE**, and **.ELSEIF**. The assembler generates the appropriate code to implement the control structure. See Chapter 7, “Controlling Program Flow.”
- MASM now has more powerful features for defining and calling procedures. The extended **PROC** syntax for generating stack frames has been enhanced since version 5.1. You can also use the **PROTO** directive to prototype a procedure, which you can then call with the **INVOKE** directive. **INVOKE** automatically generates code to pass arguments (converting them to a related type, if appropriate) and makes the call according to the specified calling convention. See Chapter 7, “Controlling Program Flow.”
- MASM optimizes jumps by automatically determining the most efficient coding for a jump and then generating the appropriate code. See Chapter 7, “Controlling Program Flow.”
- Maintaining multiple-module programs is easier in MASM 6.1 than in version 5.1. The **EXTERNDEF** and **PROTO** directives make it easy to maintain all global definitions in include files shared by all the source modules of a project. See Chapter 8, “Sharing Data and Procedures Among Modules and Libraries.”

The assembler has many new macro features that make complex macros clearer and easier to write:

- You can specify default values for macro arguments or mark arguments as required. And with the **VARARG** keyword, one parameter can accept a variable number of arguments.

- You can implement loops inside of macros in various ways. For example, the new **WHILE** directive expands the statements in a macro body while an expression is not zero.
- You can define macro functions, which return text macros. Several predefined text macros are also provided for processing strings. Macro operators and other features related to processing text macros and macro arguments have been enhanced. For more information on all these macro features, see Chapter 9, “Using Macros.”

MASM 6.1 has other improved capabilities, such as:

- The **.STARTUP** and **.EXIT** directives automatically generate appropriate startup and exit code for your assembly-language programs. See Chapter 2, “Organizing Segments.”
- MASM 6.1 supports flat memory model, available with the new Microsoft ® Windows NT ™ operating system. Flat model allows segments as large as 4 gigabytes instead of 64K (kilobytes). Offsets are 32 bits instead of 16 bits. See Chapter 2, “Organizing Segments.”
- The program H2INC.EXE converts C include files to MASM include files and translates data structures and declarations. See Chapter 20 in *Environment and Tools*.
- MASM 6.1 provides a library of assembly routines that let you create a terminate-and-stay-resident program (TSR) in a high-level language.

MASM 6.1 includes many other minor new features as well as extensive support for features of earlier versions of MASM. For a complete list of enhancements, refer to Appendix A, “Differences between MASM 6.1 and 5.1.” The cross-references in Appendix A guide you to the chapters where the new features are described in detail.

MASM Features New Since Version 6.0

MASM 6.1 offers several new features:

- ML now runs in 32-bit protected mode under MS-DOS, giving it direct access to extended memory for assembling very large source files.
- A collection of tools lets you write a dynamic-link library (DLL) for the Microsoft ® Windows ™ operating system without the Windows Software Development Kit. The LIBW.LIB library provides access to all functions in the Windows application programming interface (API), so your DLL can display menus, dialog boxes, and scroll bars. Chapter 10, “Writing a Dynamic-Link Library for Windows,” shows you how.

- Program listings now show instruction timings. The number of required processor cycles appears adjacent to each instruction in the listing, based on the selected processor. For an example listing and instructions on how to use this feature, see Appendix C, "Generating and Reading Assembly Listings."
- All utilities have been updated for version 6.1. Documentation is clearer and better arranged, with a new *Environment and Tools* reference book.
- Version 6.1 generates debugging information for CodeView version 4.0 and later.
- MASM 6.1 provides even greater compatibility with version 5.1 than does MASM 6.0. Many programs written with version 5.1 will assemble unchanged under MASM 6.1.

ML and MASM Command Lines

MASM 6.1 provides an updated version of the command-line driver, ML, introduced in version 6.0. ML is more powerful and flexible than the MASM driver of version 5.1. ML assembles and links with one command. It recognizes all the old MASM driver command syntax, however, to support existing batch files and makefiles that use MASM command lines.

Note The name MASM has traditionally referred to the Microsoft Macro Assembler. It is used in that context throughout this book. However, MASM also refers to MASM.EXE, which has been replaced by ML.EXE. In MASM 6.1, MASM.EXE is a small utility that translates command-line options to those accepted by ML.EXE, and then calls ML.EXE. The distinction between ML.EXE and MASM.EXE is made whenever necessary. Otherwise, MASM refers to the assembler and its features.

Compatibility with Earlier Versions of MASM

MASM 6.1 is fully compatible with version 6.0 and, in many cases, with version 5.1. Code written for MASM 5.1 will often assemble correctly without modification under MASM 6.1. However, MASM 6.1 provides the **OPTION** directive to let you selectively modify the assembly process. In particular, you can use the **M510** argument with **OPTION** or the /Zm command-line option to set most features to be compatible with version 5.1 code.

For information about obsolete features that will not assemble correctly under MASM 6.1, see Appendix A, "Differences Between MASM 6.1 and 5.1." The appendix also explains how to update code to use the new features.

A Word About Instruction Timings

As an assembly-language programmer, whether novice or expert, you are probably interested in producing lightning-fast code. After all, one of the main reasons to program in assembly is to take advantage of its ability to streamline execution speeds to the limit of the processor. This book will help you write efficient and fast programs.

When discussing the speed of individual instructions, the chapters in this book often speak of "timing," which is the number of processor cycles required to carry out an instruction. The *Reference* lists instruction timings for processors in the 8086 family. It is tempting to use timing as the only criterion when judging an instruction's actual execution speed, but the world within the processor is not so simple.

The clock for instruction timing does not begin ticking until the processor has read and begins to execute an instruction. When you read about instruction timings (in this book or any other), keep in mind that other factors also influence the real speed of an instruction: the instruction's size, whether it resides in cache memory, whether it accesses memory, its position in the processor's prefetch queue, and the processor type. These factors make it impossible to say precisely how fast an instruction executes. Accept the references to timing in this book as guidelines, but use these simple rules to write fast code:

- Whenever possible, use registers rather than constant values, and constant values rather than memory.
- Minimize changes in program flow.
- Smaller is often better. For example, the instructions

  ```
  dec  bx
  sub  bx, 1
  ```

  accomplish the same thing and have the same timings on 80386/486 processors. But the first instruction is 3 bytes smaller than the second, and so may reach the processor faster.
- When possible, use the string instructions described in Chapter 5, "Defining and Using Complex Data Types."

Books for Further Reading

The following books may help you learn to program in assembly language or write specialized programs. These books are listed only for your convenience. Microsoft makes no specific recommendations concerning any of these books.

Books About Programming in Assembly Language

Abrash, Michael. *Zen of Assembly Language*. Glenview, IL: Scott, Foresman and Co., 1990. Out of print.

Duntemann, Jeff. *Assembly Language from Square One: For the PC AT and Compatibles*. Glenview, IL: Scott, Foresman and Co., 1990. Out of print.

Fernandez, Judi N., and Ruth Ashley. *Assembly Language Programming for the 80386.* New York: McGraw-Hill, 1990.

Miller, Alan R. *DOS Assembly Language Programming*. San Francisco: SYBEX, 1988. Out of print.

Scanlon, Leo J. *80286 Assembly Language Programming on MS-DOS Computers*. New York: Brady Communications, 1986. Out of print.

Turley, James L. *Advanced 80386 Programming Techniques*. Berkeley, CA: Osborne McGraw-Hill, 1988.

Books About MS-DOS and BIOS

"Terminate-and-Stay-Resident Utilities." *MS-DOS Encyclopedia.* Redmond, WA: Microsoft Press, 1989.

Duncan, Ray. *Advanced MS-DOS Programming: The Microsoft Guide for Assembly Language and C Programmers*. 2d ed. Redmond, WA: Microsoft Press, 1988.

Duncan, Ray. *Extending DOS: Programmer's Guide to Protected-Mode DOS*. Redding, MA: Addison-Wesley. 1991.

Jourdain, Robert. *Programmer's Problem Solver for the IBM PC, XT and AT*. New York: Brady Communications, 1985. Out of print.

Microsoft MS-DOS Programmer's Reference. Redmond, WA: Microsoft Press, 1991.

Norton, Peter and Richard Wilton. *The New Peter Norton Programmer's Guide to the IBM PC and PS/2*. Redmond, WA: Microsoft Press, 1988.

Wilton, Richard. *Programmer's Guide to PC & PS/2 Video Systems: Maximum Video Performance from the EGA, VGA, HGC, and MCGA*. Redmond, WA: Microsoft Press, 1987. Out of print.

Books and Articles About Windows

Kauler, Barry. *Windows Assembly Language & Systems Programming: Object-Oriented & Systems Programming in Assembly Language for Windows 3.0 and 3.1.* New York, NY: Prentice Hall, 1993.

Klein, Mike. *Windows Programmer's Guide to DLLs & Memory Management*. Carmel, IN: Sams, 1992.

Petzold, Charles. *Programming Windows*. 3d ed. Redmond, WA: Microsoft Press, 1992.

Petzold, Charles. "Environments." *PC Magazine*. New York, NY: Ziff-Davis Publishing Company, June 1990–1992.

Programmer's Reference. 4 vols. Microsoft Windows Software Development Kit (SDK). Redmond, WA: Microsoft Press, 1992.

Books About Other Topics

Nelson, Ross P. *The 80386/80486 Programming Guide*. 2d ed. Redmond, WA: Microsoft Press, 1991.

Startz, Richard. *8087/80287/80387 for the IBM PC and Compatibles: Applications and Programming with Intel's Math Coprocessors*. Bowie, MD: Robert J. Brady Co., 1988. Out of print.

Document Conventions

The following document conventions are used throughout this manual:

Example of Convention	**Description**
SAMPLE2.ASM	Uppercase letters indicate filenames, segment names, registers, and terms used at the command level.
.MODEL	Boldface type indicates assembly-language directives, instructions, type specifiers, and predefined macros, as well as keywords in other programming languages.
placeholder	Italic letters indicate placeholders for information you must supply, such as a filename. Italics are used occasionally for emphasis in the text.
`target`	This font is used to indicate example programs, user input, and screen output.
`;`	A semicolon in the first column of an example signals illegal code. A semicolon also marks a comment.

SHIFT	Small capital letters signify names of keys on the keyboard. Notice that a plus (+) indicates a combination of keys. For example, CTRL+E means to hold down the CTRL key while pressing the E key.
⟦*argument*⟧	Items inside double square brackets are optional.
{*register*|*memory*}	Braces and a vertical bar indicate a choice between two or more items. You must choose one of the items unless double square brackets surround the braces.
Repeating elements...	A horizontal ellipsis (...) following an item indicates that more items having the same form may appear.

`Program`
 `.`
 `.`
 `.`
 `Fragment`
	A vertical ellipsis tells you that part of a program has been intentionally omitted.

Getting Assistance and Reporting Problems

If you need help or think you have discovered a problem in the software, please provide the following information to help us locate the source of the problem:

- The version of MS-DOS or Windows you run.
- Your system configuration: the type of machine you use, its total memory, and its total free memory at assembler execution time, as well as any other information you think might be useful.
- The command line you used for the assembler, linker, or other MASM tool that was running when the problem occurred.
- Any object files or libraries you linked with if the problem occurred at link time.

If your program is very large, reduce it to the smallest possible program that still produces the problem.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the section "Microsoft Support Services" in the introduction to *Environment and Tools*. If you have comments or suggestions regarding any of the books accompanying this product, please indicate them on the Document Feedback page at the back of this book and send it to Microsoft.

If you have not yet registered your copy of the Macro Assembler, you should fill out and return the Registration Card. This enables Microsoft to keep you informed of updates and other information about the assembler.