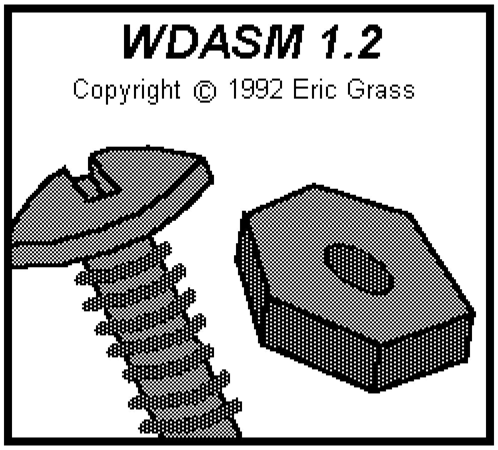
WDASM
Windows Disassembler 1.2



User's Manual

Index

Version 1.2 of this software is shareware, user supported software that is distributed to the user without cost. You are allowed to evaluate this product on your own system on a trial basis. If you find this product to be of value and intend to continue using it, then you are asked to register your copy and pay the license fee. Registered users of *WDASM* 1.0 and 1.1 are entitled to a free upgrade to version 1.2. Commercial and/or business use of *WDASM* by non-registered users is prohibited.

Introduction

Windows Disassembler is created specifically for disassembling Windows executables and dynamic link libraries. It allows you to browse at the assembly language code of a program without having to write it to a file. *WDASM* generates procedure directives, as well as all of the literal Windows API function call names.

Specifications

Files
Works on Windows 3.x executables and dynamic link libraries only.
Instruction Set
Translates all instructions within the 286 instruction set with the exception of the following multi-tasking instructions: LAR, LGDT, LIDT, LLDT, LMSW, LSL, LTR, SGDT, SIDT, SLDT, SMSW, STR, VERR, and VERW.
Operating System and Hardware
Requires at least DOS 4.0, Windows 3.0, and a 286 or above IBM compatible computer. Installation of *SMARTDRV* (which comes with Windows) is recommended.

Operation

Opening Files
The default extension if you do not specify one is "**.EXE**". Windows Disassembler views one file at a time. If you open a file while another one is already open, the old file will be automatically closed. When opened, the file's assembly language code is appears on the screen, provided that the file has a DOS executable file header, a new executable file header, and at least one segment. Otherwise, a dialog box will inform you that the file does not meet a particular specification.

Viewing Assembly Language Code
Viewing code in the display window is presented as an alternative to generating a gigantic assembly language source code file, since some programs are bound to be quite large, and you may merely want to browse at a program's machine code.

The code that initially appears in the window when a file is opened is the first segment within the file. Numbers are assigned to segments according to their chronological order within the new executable file header. Windows Disassembler displays one segment at a time within the window. The **View | Segment** command must be used to go to another segment. To scroll the text in the window, use the Up Arrow, Down Arrow, Page Up, and Page Down keys. Alternatively, of course, you may use the scroll bar. To see the address offsets of each instruction, select **View | Address Offsets** from the main menu. To jump to any address location in the segment, select the **View | Go To** command, or use the scroll bar thumb. The **Go To** command asks you for a hexadecimal address.

New to versions 1.1 and 1.2 is the **View | Far Call Names** command. This toggles between viewing far function call names and the actual relocation values in **CALL** instructions (for example, you will typically see the numbers **0000H:0FFFFH**). NOTE: This setting WILL affect the way the code is written to a file.

All labels take form of **LxxxxH** and **DxxxxH**, where **xxxx** is a 4-digit hexadecimal number equal to the offset of the location being referenced. Labels with an '**L**' prefix denote locations within the immediate code segment, and labels with a '**D**' prefix denote locations within a data segment. Labels within a code segment can either be procedure labels, jump/loop labels, or data labels within the code segment. Assembler directives, while generated for source code text files, are not shown in the display window.

Strings are detected and translated by *WDASM* whenever five or more visible characters occur within a data segment.

The **Set Byte** command allows you to convert a desired range of bytes from byte declarations into instructions, or vice versa, or to give labels to the specified range of bytes. This command is necessary for programs which have data declarations in their code segments. ONE WORD OF CAUTION: All byte settings which you have made in a segment will be lost when you exit a segment. You can save that segment (with the **Save Current Segment Only** option only) in a text file first before quitting to save the changes. But if you quit the program and restart it, there is no way to restore the previous byte settings except by doing it over again manually. If you use the **Create Separate Files For Each Segment** option, you will lose your changes as well.

Generating An Assembly Language Source Code File

After you open a file, you can write the assembly language source code of that file to a new text file. If the file name you specify is one that already exists, the specified file will be automatically overwritten with a new one. Three options are available for generating a file(s). The first is to put all of the executable code into one file. The name of this file will be the name you specify. The second option is to put each segment of the code into separate files. Each segment's file name will be of the form ***yourname.nnn***, where ***yourname*** is the name you specify in the dialog box, and ***nnn*** is an integer corresponding to the segment's number. For example, if you specify **\work\myprog.asm** as the file name, Windows Disassembler will generate files named **\work\myprog.1**, **\work\myprog.2**, **\work\myprog.3**, etc.. The third option is to generate a file for the current segment only (which is currently being displayed on the screen). The file name will have an integer file extension also.

All byte settings made with the **Set Byte** will be lost if you exit a given segment, or if you try writing all of the segments to a file(s) at one time. However, if you save a single segment in a text file using the **Save Current Segment Only** option, all byte settings will stay safe and will still be there after saving the segment.

The new file will contain tabs. To view the file in the way in which it was intended to be viewed, you should set your editor's tab stop value to 8 spaces.

WDASM will automatically generate **TITLE**, **.CODE**, **.DATA <*segmentname*>**, **.MODEL LARGE**, **.286**, and **EXTRN <*winAPIfunc*>:FAR** directives in the file. A relatively new feature is the automatic generation of **PROC** and **ENDP** directives for all exported and far procedures. These directives will take the following form in the new source code:

```
Proceduren    PROC  FAR PUBLIC
              (code)
              RETF
Proceduren    ENDP
```

where ***n*** is the ordinal number of the procedure in the entry table of the program's executable file header. Moreover, for far calls to procedures within the program in a different segment, **EXTERNDEF**'s are generated.

WDASM generates segment names of the form **.CODE SEG*n***, where ***n*** is the segment number. In some cases, you'll end up deleting the **SEG*n***. This name is produced in order to help you identify a segment. However, if your segments have been written to separate files then the name is optional (see the *MASM* programmer's guide). *WDASM* assumes the program had a large memory model. If this assumption is wrong then you must delete one or two segment names. Windows Dissassembler 1.2 translates functions belonging to **COMMDLG.DLL** and **SHELL.DLG**, and generates useful information for unknown function calls. This information is in the form **Module *m* Ordinal *n***. For example, the occurance of an unknown far function call will cause an "**Module *m* Ordinal *n*:FAR**" statement to appear at the beginning of the file, plus one or more "**CALL Module *m* Ordinal *n*:FAR**" statements in the code. You must look up these function names using an executable-file header utility. (In other words, you must use the relocation table names and offsets provided by an **.EXE** file header utility to determine the far function call/variable names in the assembly code.)

Finally, you must figure out the entry point (used by the **END** directive) using a **.EXE** file header utility as well, plus make sure you have the appropriate **.MODEL <*type*>** directive and add some **EXTRN**'s (or **EXTERNDEF**'s) for any far variables used by the program (typically the far variable **__winflags** is used by Windows programs, for example).

As an example, the files **HELLO.EXE**, **HELLO.C**, **HELLO.DEF**, **HELLO.EXH**, and **HELLO2.ASM** are included to demonstrate disassembly using *WDASM*. **HELLO.EXE** (a "hello world" program) is a compilation of **HELLO.C** with *Microsoft QuickC for Windows*. **HELLO.EXH** is an **.EXE** file-header listing for **HELLO.EXE** generated by *EXEHDR*. **HELLO2.ASM** was generated using *WDASM* and was edited in order to change all occurances of **Procedure1** to **WndProc**, as well as to provide the symbols **OFFSET WndProc**, **SEG WndProc**, **START** (the entry point), and **OFFSET __WINFLAGS** (using the information in an **.EXE** header file listing for **HELLO.EXE**). Also, the model type was changed to **SMALL**, an **EXTRN __WINFLAGS** directive was added and the segment names **SEG1** and **SEG2** were deleted. You can rebuild **HELLO.EXE** from **HELLO2.ASM** with *MASM 6.0* by typing:

```
ML /c HELLO2.ASM
LINK /ALIGN:2 HELLO2,,HELLO2, libw slibcew, hello.def;
```

which will generate **HELLO2.EXE.**

Re-assembling medium, compact, and large model programs is slightly more complex than the example just given. Without going into much detail, the simplest way to disassemble and reassemble a medium/large-model program is to first save the segments in separate files (or modules). Then, in addition to the steps described above, do the following. Make your data segment accessible to all modules by copying the contents of the data segment file to a new file and converting it into an include file. This is done using an editor with regular expression finding/replacing capabilities and replacing each occurance of "**D*xxxx*H DB *nnnn*H**" with an "**EXTERNDEF D*xxxx*H:BYTE**" and then saving the file with an **.INC** extension. Then include this file (i.e., **INCLUDE *filebasename*.INC**) in each module that accesses the data segment. (If there are two data segments, then there could be conflicting labels.) Finally, assuming you've got the resource files, assemble each module and link. Otherwise, Borland's *Resource Workshop* can be used for obtaining the resources, or any reverse resource compiler.

Differences Between Windows Disassembler 1.2 and 1.1

New Features

Version 1.2 enables the user to convert any range of bytes from data bytes to instruction bytes and vice-versa before generating a file. In addition, *WDASM* now generates **FAR PTR**'s, **NEAR PTR**'s, and **SHORT** directives for **CALL** and **JMP** instructions, in order to reproduce a program more precisely. **WORD PTR**'s are also now provided for 16-bit displacement bytes inside the brackets of indirect memory operands for the same purpose. Plus, the **.MODEL** directive now has a default type of **MEDIUM** instead of **LARGE**, and the **END** directive is now supplied.

NOTE: A reminder message appearing every 4 minutes has been added as an extra incentive for registering (see **Registration** form below).

Bug Fixes

(Bug fixes for version 1.1 are included here in addition to those for 1.2.)

Version 1.1: A bug concerning the translation of the **LES** instruction was corrected (version 1.0 accidently skipped the two bytes which came after an **LES**). In addition, a bug previously existed concerning the calculation of an API module's ordinal number. The effect was that function call names weren't provided for some programs. Finally, *WDASM*'s ability to detect null bytes - bytes having a value of zero which are usually stuck in the code before a procedure - was expanded to include null bytes occuring between far calls and **PUSH BP** instructions.

Version 1.2: Bugs in the translation of **PUSHF** and **BOUND** have been corrected. (The 'F' was missing at the end of "PUSHF" and "BOUND" was misspelled as "UN D".)

A Slight Bug In This Version

The screen will need refreshing sometimes after scrolling upwards, mainly within data segments, but sometimes in code segments if you set some of the byte types. This bug is minor and will not affect file generation.

Registration

The single license fee for *WDASM* version 1.2 is only $10.00. This version, version 1.2, contains a reminder message for registering the program which appears at 4 minute intervals. Registering this program entitles you to receive the actual version of this program without the reminder message. Please fill out this form (or a reasonable facsimile thereof) and send it with your check or money order for $10.00 to:

Eric Grass **(314) 928-7803**
1612 Gettysburg Landing
St. Charles, MO 63303

Date ___________________

Name ____ Phone __________________

Address ____

City ____________________________________ State ________ Zip _____________

Please indicate which type of disk you use:

_______ 5.25" ________ 3.5"

Product:	*WDASM* Windows Disassembler 1.2
Total Price:	$10.00

Please make your check or money order payable to Eric Grass.

Comments, critiques and suggestions regarding Windows Disassembler 1.2 are welcomed and can be forwarded to the above address.

License

Warranty Disclaimer

Copyright