This document lists some of the differences between Asmc, JWasm, and Masm.

In This Section

Asmc Command-Line Option

Describes the Asmc command-line option.

Asmc Error Messages

Describes Asmc fatal and nonfatal error messages and warnings.

Asmc Extensions

Provides links to topics discussing Masm versus Asmc.

Directives Reference

Provides links to topics discussing the use of directives in Asmc.

Symbols Reference

Provides links to topics discussing the use of symbols in Asmc.

Change Log | Forum

Asmc Command-Line Reference

Assembles and links one or more assembly-language source files. The command-line options are case sensitive.

ASMC [[options]] filename [[[[options]] filename]]

options

The options listed in the following table.

Set CPU: 0=8086 (default), 1=80186, 2=80286,

3=80386, 4=80486,

/[0|1|..|10][p] 5=00300, 4=00400, 5=Pentium,6=PPro,7=P2,8=P3,9=P4,10=x86-64. [p]

allows privileged instructions.

/assert Generate .assert(code). Same as .assert:on.

/bin Generate plain binary file.

Push user registers before <u>stack-frame</u> is created in a

proc.

/**coff** Generate COFF format object file.
/**Cp** Preserves case of all user identifiers.

/Cu Maps all identifiers to upper case (default).

/cui Link switch used with /pe -- subsystem:console

(default).

/Cx Preserves case in public and extern symbols.

Defines a text macro with the given name. If value is

/**D**symbol[[=value]] missing, it is blank. Multiple tokens separated by

spaces must be enclosed in quotation marks.

/**enumber** Set error limit number.

/elf Generate 32-bit ELF object file.
/elf64 Generate 64-bit ELF object file.

/**EP** Generates a preprocessed source listing (sent to

STDOUT). See /Sf.

/**eq** Don't display error messages. /**Fd[***file***]** Write import definition file.

/Fifile Force *file* to be included.

/**Fl**[[*filename*]] Generates an assembled code listing. See /Sf.

/**Fofilename** Names an object file. /**Fwfilename** Set errors file name.

/**FPi** Generates emulator fix-ups for floating-point

arithmetic (mixed language only).

/**FPi87** 80x87 instructions (default).

/**fpc** Disallow floating-point instructions. /**fpn** Set FPU: 0=8087, 2=80287, 3=80387.

Specifies use of FORTRAN- or Pascal-style function

/Gc calling and naming conventions. Same as **OPTION**

LANGUAGE:PASCAL.

/**Gd** Specifies use of C-style function calling and naming

conventions. Same as **OPTION LANGUAGE:C**.

/gui Link switch used with /pe -- subsystem:windows.

/Gv Specifies use of VECTORCALL-style function calling

and naming conventions.

Specifies use of STDCALL-style function calling and

/Gz naming conventions. Same as **OPTION**

LANGUAGE:STDCALL.

Copy Register Parameters to Stack. Forces parameters

/homeparams passed in registers to be written to their locations on

the stack upon function entry.

/**Ipathname** Sets path for include file.

 $/\mathbf{m}[t|s|c|m|l|h|f]$ Set memory model.

/mz Generate DOS MZ binary file.
/ncname Set class name of code segment.

/**nd***name* Set name of data segment.

/nm*name* Set name of module.

/**ntname** Set name of text segment.

/nologo Suppresses messages for successful assembly.

/omf Generates object module file format (OMF) type of

object module.

/**pe** Generate PE binary file, 32/64-bit.

/**pf** Preserve Flags (Epilogue/Invoke).

/q Suppress copyright message.

/**r** Recurse subdirectories with use of wild args. /**Sa** Turns on listing of all available information.

Marks the object as either containing no exception

/safeseh handlers or containing exception handlers that are all

declared with SAFESEH.

/**Sf** Adds first-pass listing to listing file.

/**Sg** Turns on listing of assembly-generated code.
/**Sn** Turns off symbol table when producing a listing.

/**Sp[n]** Set segment alignment.

Specifies use of <u>C-style</u> .SWITCH convention

(default).

/**swn** No jump-table creation in <u>.SWITCH</u>.

/swp Specifies use of Pascal-style .SWITCH convention

(auto break).

/swr Allows use of register [R|E]AX in .SWITCH code.

/swt Allows use of jump-table creation in .SWITCH code

(default).

/Sx Turns on false conditionals in listing.

/w Same as /W0.

/swc

/W*level* Sets the warning level, where level = 0, 1, 2, or 3.

/win64 Generate 64-bit COFF object.

/ws[CodePage] Store quoted strings as Unicode. See OPTION

WSTRING.

/WX Returns an error code if warnings are generated.

/X Ignore INCLUDE environment path.

/**Xc** Disable <u>Asmc extensions</u>

/zcw No decoration for C symbols.

Zd Generates line-number information in object file.

/Zf Make all symbols public.

/**zf[0|1]** Set FASTCALL type: MS/OW. /**Zg** Generate code to match Masm.

/**Zi[0**|**1**|**2**|**3**] Add symbolic debug info.

/**zlc** No OMF records of data in code.

/**zld** No OMF records of far call.

/zlf Suppress items in COFF: No file entry./zlp Suppress items in COFF: No static procs.

/**zls** Suppress items in COFF: No section aux entry.

/**Zm** Enable MASM 5.10 compatibility. /**Zne** Disable non Masm extensions.

/Zp[[alignment]] Packs structures on the specified byte boundary.

/**Zs** Perform syntax check only. /**zt**<**0**|**1**|**2**> Set STDCALL decoration.

/Zv8 Enable Masm v8+ PROC visibility.

/zze No export symbol decoration.
/zzs Store name of start address.

filename

The name of the file.

Environment Variables

INCLUDE Specifies search path for include files.

ASMC Specifies default command-line

options.

TEMP Specifies path for temporary files.

See Also

ASMC Error Messages

Asmc Error Messages

The error messages generated by Asmc components fall into three categories:

- **Fatal errors (A1000-A1901):** These indicate a severe problem that prevents the utility from completing its normal process.
- **Nonfatal errors (A2000-A3021):**

The utility may complete its process. If it does, its result is not likely to be the one you want.

• Warnings (A4000-A8020):

These messages indicate conditions that may prevent you from getting the results you want.

All error messages take the following form:

```
Utility: Filename (Line) : [Error_type] (Code): Message_text
```

Utility

The program that sent the error message.

Filename

The file that contains the error-generating condition.

Line

The approximate line where the error condition exists.

Error_type

Fatal Error, Error, or Warning.

Code

The unique 5- or 6-digit error code.

Message_text

A short and general description of the error condition.

See Also

Asmc Extensions

The main goal with Asmc is an attempt to provide more readability to the assembly language based on Masm syntax but at the same time keep compatibility with existing source code. In order to achieve this some of the main core of the assembler has to be rewritten and the HLL section enhanced as discussed below.

In This Section

Parsing of labels

Provides links to topics discussing parsing of labels.

Expansion of macros

Provides links to topics discussing expansion of macros.

The invoke directive

Provides links to topics discussing the invoke directive.

The HLL section

Provides links to topics discussing the HLL section.

Handling of strings

Provides links to topics discussing handling of strings.

Enhanced vector extension

Provides links to topics discussing Enhanced vector extension.

See Also

Directives Reference

OPTION optionlist

Enables and disables features of the assembler. Added options by ASMC include:

- OPTION ASMC
- OPTION CSTACK
- OPTION SWITCH
- OPTION LOOPALIGN
- OPTION CASEALIGN
- OPTION WSTRING

Conditional Control Flow

- <u>.IF</u>
- .WHILE
- .REPEAT
- .UNTILCXZ
- .BREAK
- .CONTINUE
- .FOR
- .SWITCH
- .CASE
- .ENDC
- <u>.GOTOSW</u>
- .DEFAULT
- <u>.ENDSW</u>
- .COMDEF
- .CLASSDEF
- .ENDS
- .ASSERT
- Flag conditions
- Signed compare
- Return code

See Also

Symbols Reference

Symbols Reference

Date and Time Information

• <u>@Date</u>

Macro Functions

• @CStr

See Also

Directives Reference

0. Contents

1. About 2. Commandline Options Options -0, -1, -2, ..., -10 Option -bin Option -coff Option -djgpp Option -e Option -elf Option -elf64 Option -eq Option -Fd Option -FPi Option -Fw Options -Gc, -Gd, -Gr, -Gz Option -m Option -mz Option -nc Options -nd, -nt Option -nm Option -pe Option -win64 Option -zc Option -Zd Option -zf Option -Zg Option -Zi Options -zlc and -zld Options -zlf, -zlp and zls Option -Zm Option -Zne Option -zt Option -Zv8 Option -zze

Option -zzs

- 3. **Syntax Extensions**
- 3.1 <u>Directive INCBIN</u>
- 3.2 FASTCALL Register Calling Convention
- 3.3 IDs enclosed in Back Quotes
- 3.4 Floating-Point Immediates in Instructions
- 3.5 <u>Directive OPTION FIELDALIGN</u>
- 3.6 Directive OPTION PROCALIGN
- 3.7 Directive OPTION MZ
- 3.8 Directive OPTION ELF
- 3.9 Directive OPTION WIN64
- 3.10 <u>Directive OPTION FRAME</u>
- 3.11 Directive OPTION RENAMEKEYWORD
- 3.12 Directive OPTION DLLIMPORT
- 3.13 <u>Directive OPTION CODEVIEW</u>
- 3.14 Directive OPTION STACKBASE
- 3.15 Directives PUSHCONTEXT / POPCONTEXT ALIGNMENT
- 3.16 Directives .X64 and .X64p
- 3.17 Attribute LABEL for first Macro Parameter
- 3.18 Member Argument for IF[N]DEF and .ERR[N]DEF
- 3.19 Initialization of Data Items of Type MMWORD/XMMWORD
- 3.20 Optional Array Size for LABEL Directive
- 3.21 Optional Name Argument for Simplified Segment Directives
- 3.22 <u>Segment Attribute COMDAT</u>
- 3.23 Attribute VARARGML for last Macro Parameter
- 3.24 Miscellaneous
- 3.24.1 Numeric constant JWASM
- 3.24.2 Operating System Argument for .MODEL
- 3.24.3 <u>Accepted Parameters for IF[N]DEF Directive</u>
- 3.24.4 Visibility of Procedures
- 3.24.5 Non-RIP-Relative Addressing in 64-Bit
- 4. <u>Instruction Sets</u>
- 5. Code Generation Differences
- 5.1 Forward References in Macro Expressions
- 6. Output Formats
- 6.1 OMF Output Format
- 6.2 COFF Output Format
- 6.3 Win64 Output Format
- 6.4 Binary Output Format
- 6.5 PE Output Format

- 7. Masm Bugs fixed in JWasm
- 8. Optional Features
- 9. Known Bugs and missing Features
- 10. License
- Appendix A. Reserved Words
- Appendix B. Source Samples
- Appendix C. Errors and Warnings
- Appendix D. <u>Differences between Masm 6 and Masm 8</u>
- Appendix E. Restrictions of precommpiled 16-bit Binary JWASMR.EXE
- Appendix F. Additional Features of JWasm's Debug Version

1. About

This document lists the differences between JWasm and Masm, as far as the user interface is concerned.

When Masm is mentioned, then usually Masm v8.00 is meant, unless stated otherwise. Masm v8.00 also was the first Masm version supporting 64-bit (ML64.EXE).

The Masm documentation itself can be found on numerous places in the web, in plain text, HTML, PDF and Windows Help format. However, it's usually just the documentation that came whith Masm v6.1 - hence a bit outdated nowadays (the changes from Masm v6 to Masm v8 are listed in $\underline{\text{Appendix D}}$).

2. Commandline Options

Entering 'JWasm -?' or 'JWasm -h' will make JWasm display the options it understands. A lot of them exist in both Masm and JWasm, but some are valid for JWasm only; OTOH, a few options are missing in JWasm.

Options are usually entered via the command line. Additionally, when starting, JWasm will search for **environment variable JWASM** and handle it similar to the way Masm handles variable ML. Hence it is also possible to enter options via this method.

The options specific to JWasm - and also the options which are handled somewhat differently by JWasm compared to Masm - will be handled in the following chapters.

One major difference should be mentioned here: JWasm does never launch a linker on its own (which Masm does unless option -c is given). This makes a few Masm options useless for JWasm, and hence they are not implemented; see Chapter Known Bugs and missing Features for details.

Options -0, -1, -2, ..., -10: Select Cpu

Option [0|1|..|10] selects cpu/instruction set. Most values correspond to cpu directives:

0	.8086
1	.186
2	.286
3	.386
4	.486
5	.586
6	.686
7	.686 and .MMX (P2)
8	.686, .MMX and SSE instructions (P3)
9	.686, .MMX, SSE and SSE2 instructions (P4)
10	.x64 (x86-64 cpu)

Option -bin: Select Output Format Binary

Option -bin selects output format BINary. The output module's default file extension will be changed from .OBJ to .BIN.

For more information, see **Binary Output Format**.

Option -coff: Select Output Format COFF

Option -coff selects output format COFF. The most common use for -coff is to produce modules for 32-bit Windows. However, it's not mandatory that the output will be 32-bit with -coff; depending on what cpu is selected when the .MODEL directive will be parsed, the output may be 32- or 64-bit.

For more information about COFF, see **COFF Output Format**.

Option -djgpp: Select Output Format for DJGPP (optional)

Option -djgpp selects Djgpp's variant of COFF as output format. Since it is rarely used nowadays, this option isn't activated in the precompiled binaries. See Optional Features how to enable it.

Option -e: Set error limit

Allows to set the number of errors after which the assembly process is aborted. The default value is 50.

Option -elf: Select 32-bit Output Format ELF

Option -elf selects output format ELF. JWasm will produce an object module in 32-bit Elf format.

Use **OPTION ELF** to set values in the ELF header.

Option -elf64: Select 64-bit Output Format ELF

Option -elf64 selects output format ELF64. JWasm will produce an object module in 64-bit Elf format. Additionally, this option will set cpu to <u>x86-64</u> and model to FLAT.

Use **OPTION ELF** to set values in the ELF header.

Option -eq: Suppress Error Messages on Screen

Option -eq will suppress displaying error messages on the screen. They are still written into an error file.

Option -Fd: Write Import Definitions

Option -Fd makes JWasm write import definitions in a format understood by Open Watcom's Wlink and JWlink. Such definitions will tell the linker how to resolve the external reference; no import library is needed. This option is only useful in conjunction with <u>OPTION DLLIMPORT</u> (chapter 3.12). Syntax is:

-Fd[=file_name]

If the optional <file_name> argument is given, the import definitions will be written into a file of this name. This is the only way for Open Watcom's Wlink to pass the information.

If JWlink is used, the <file_name> argument may be omitted. Then JWasm will write the import definitions directly into the object module's linker directive section (section ".drectve"). This works for output formats COFF and ELF only. See sample Win32_7 how to use JWasm and JWlink to create a Windows binary without import libs.

Option -FPi: Activate Floating-Point Emulation

Option -FPi activates "inline FP instructions with emulation". This will make JWasm create fixups for floating-point instructions if code is 16bit. If supported by the linker or the OS, the FP instructions can then be replaced by calls to an FP emulator if no coprocessor exists.

Option -Fw: Set Error File Name

Option -Fw will set the file name for warning and error messages. As default, these messages are written to a file with a name equal to the assembly source, but with extension .ERR. Syntax is:

-Fw file_name

Options -Gc, -Gd, -Gr, -Gz: Set Default Calling Convention

The default calling convention set by these options are:

- -Gc Pascal
- -Gd C(decl)
- -Gr Fastcall
- -Gz Stdcall

Option -m: Select Memory Model

Option -m generates a line containing a .MODEL directive to select a memory model. Syntax is:

```
-m[t|s|m|c|l|h|f]
```

where the value behind 'm' means:

```
t = tiny
s = small
m = medium
c = compact
l = large
h = huge
f = flat
```

This option is ignored if a 64-bit output format (<u>-win64</u> or <u>-elf64</u>) is active.

Option -mz: Select Output Format MZ

Option -mz selects output format MZ. This will write a binary in DOS MZ format. The module's default file extension will be changed from .OBJ to .EXE. All symbols in the module must resolve internally, no externals are allowed. Some values in the "MZ" header can be adjusted by directive OPTION MZ (see below).

As in all binary formats, the listing will contain a binary map; see **Binary Output** Format for more details.

Option -nc: Set Code Class Name

Option -nc sets the code segment's class name if simplified segment directives are used (default:CODE). Syntax is:

-nc=name

where <name> will be the code segment's class name.

Options -nd and -nt: Set DATA and CODE Segment Names

Options -nd and -nt will set the name of the data/code segments if simplified segment directives are used. Syntax is:

```
-nt=name_of_code
-nd=name_of_data
```

The default names are _TEXT for code and _DATA for data.

Option -nm: Set Module Name

Option -nm sets the module name. Syntax is:

-nm=module_name

The default value for <module_name> is the name of the source file without extension.

Option -pe: Create a PE Binary

Option -pe will make JWasm create a binary in Windows PE format (32- and 64-bit). The output module's default file extension will be changed from .OBJ to .EXE.

For more details about the PE format see PE Output Format.

Chapter Output Formats lists all available output formats.

Option -win64: Select Output Format Win64

Option -win64 makes JWasm produce an object module in PE32+ format, the 64-bit format used for Win64 binaries.

Option -win64 will also set cpu to <u>x86-64</u>, model to FLAT and default calling convention to <u>FASTCALL</u>. This is to make JWasm compatible with Masm64 (ML64.EXE).

With OPTION WIN64, parameters specific to Win64 may be set.

For more information about Win64, see Win64 Output Format.

Chapter Output Formats lists all available output formats.

Option -zc: Set Name Decoration for C

This option sets the name decoration for the C (aka CDECL) calling convention. There are two variants:

-zc[m|w]

-zcm is the default, C names are decorated with an underscore prefix. This is also the way Masm does it. The other variant, -zcw, omits the prefix altogether. This is the usual "decoration" in Unix/Linux.

Option -Zd: Emit Line Number Debugging Info

Option -Zd generates line number debug information for OMF and COFF output format. For other formats, this option is ignored. Line number information allows a debugger to trace the binary on the source code level. Debuggers that have been verified to work with this option:

MS CodeView	OMF	16-bit, Windows and DOS
CDB, NTSD	COFF	32- and 64-bit, Windows
WinDbg	COFF	32- and 64-bit, Windows
MS Visual Studio 2008	COFF	32-bit ^[1] , Windows
MS Visual Studio 2010	COFF	32-bit ^[1] , Windows
OW WD/WDW	OMF/COFF	16- and 32-bit, Windows and DOS
PellesC IDE	COFF	32-bit ^[1] , Windows
Borland TD/TDW/TD32	OMF	16- and 32-bit, Windows and DOS

[1]: the 64-bit version of this software should also be able to debug 64-bit debuggees.

Option -zf: Select FASTCALL Type

Option -zf selects the FASTCALL calling convention type for 16- and 32-bit code. Syntax is:

-zf[0|1]

The default value 0 is MS VC style, while value 1 activates the Open Watcom fastcall type.

Option -Zg: Masm-compatible Code Generation

Option -Zg makes JWasm try an exact copy of Masm's code generation, which results in the following changes:

the default prologue for procedures will use

```
add [e]sp, - localsize
instead of
sub [e]sp, localsize.
```

- the default epilogue for procedures will almost always prefer to generate opcode **LEAVE** if possible.
- expressions

```
<reg> == 0 and <reg> != 0
will generate code
  or <reg>, <reg>
instead of
  cmp <reg>, 0.
```

- if invoke must extend an argument's size from word to dword, register eax remains unchanged.
- the segment value of FAR externals will always be assumed unknown, no matter whether the external is defined inside or outside of a segment block. Thus those labels are excluded from "far call optimization".
- if indirect addressing uses 2 (32- or 64-bit) registers **and** no scaling factor is used **and** the second register isn't ESP/RSP, then the registers are swapped: the second becomes base and the first becomes index (the registers in an expression are counted from left to right). [since v2.10, this is standard behavior].

Option -Zi: Emit Symbolic Debugging Info

Option -Zi generates symbolic debugging info in CodeView V4 style for OMF and COFF output format. For other formats, this option is accepted, but ignored. Note that -Zi will always enable <u>-Zd</u> (line number information). Debuggers which have been verified to work with this option:

MS CodeView	16-bit, Windows and DOS
MS CDB or NTSD	32- and 64-bit, Windows
MS WinDbg	32- and 64-bit, Windows
MS Visual Studio 2008	32-bit ^[1] , Windows
MS Visual Studio 2010	32-bit ^[1] , Windows
Open Watcom WD/WDW	16- and 32-bit, Windows and DOS
Pelles C IDE	32-bit ^[1] , Windows

[1]: the 64-bit version of this software should also be able to debug 64-bit debuggees.

Usually both the assembler and the linker must be told that symbolic debugging information is to be generated (with MS link, the linker option is /DEBUG).

The -Zi option accepts an **optional numeric argument** to control the volume of information that is emitted. The values currently accepted are:

0 Just global symbols will be written.

Global & local symbols will be written. No user-defined types are included.

- 1 Usually this reduction does no harm, but may decrease linking time if lots of modules are to be linked, the effect may be quite significant.
- 2 This is the default. Global & local symbols and user-defined types are written.
- 3 Additionally to 2, symbolic constants (equates) will be written.

Also see <u>OPTION CODEVIEW</u> for additional switches to control symbolic debugging output.

Options -zlc and -zld: Control Content of OMF Output Module

Options -zlc and -zld do reduce size of the OMF output module. They might be useful if lots of - small - modules are to be assembled and put into a static library. Also, the OMF coment records written if -zlc or -zld is NOT set may not be accepted by all linkers.

Option -zlc:

suppresses writing OMF coment records about data in Code segments. These records may help a disassembler to produce nicer listings.

Option -zld:

suppresses writing an OMF coment record for each Code segment telling the linker that far calls to targets in the same segments should be optimized. This is more or less a feature for 16-bit code only.

Options -zlf, -zlp and -zls: Control Content of COFF Output Module

Options -zlf, -zlp and -zls do reduce size of the COFF output module. They might be useful if lots of - small - modules are to be assembled and put into a static library.

Option	Meaning
-zlf	suppresses the @file entry in the COFF symbol table. This entry is usually used for debugging purposes only and hence a - pretty small - amount of space can be saved.
-zlp	suppresses static (=private) procedures to be included into the COFF symbol table - as long as they aren't referenced absolutely. Such procedures aren't needed for the linking process. However, since the linker has no knowledge of them then, they will also disappear from the linker-generated map-file.
-zls	suppresses the auxiliary entries for sections in the COFF symbol table. These entries may not be needed in all cases and thus a little space is saved.

Option -Zm: Enable Masm v5 Compatibility

Option -Zm (or setting OPTION M510) will do:

- set OPTION OLDSTRUCTS
- set OPTION DOTNAME
- set OPTION SETIF2:TRUE
- set OPTION OFFSET:SEGMENT (if no model is set)
- set OPTION NOSCOPED (if no model with language specifier is set)
- allow to define data items behind code labels
- allow "invalid" use of REP/REPE/REPNE instruction prefixes
- change precedence of [] and () operator from 1 to 9. Hence expression
- -5[bx] is parsed as (-5)[bx], while without -Zm it is parsed as -(5[bx]), which generates an error.

Other Masm v5.1 compatibility options aren't implemented yet.

Option -Zne: Disable JWasm Syntax Extensions

Option -Zne will disable syntax extensions which aren't supported by Masm. Currently these are:

- directive **INCBIN**
- calling convention **FASTCALL**
- IDs enclosed in backquotes
- floating-point immediate operands in instructions
- directive OPTION FIELDALIGN
- directive OPTION PROCALIGN
- directive OPTION MZ
- directive OPTION ELF
- directive OPTION WIN64
- directive OPTION FRAME
- directive OPTION RENAMEKEYWORD
- directive OPTION DLLIMPORT
- directive OPTION CODEVIEW
- directive OPTION STACKBASE
- directives PUSHCONTEXT / POPCONTEXT ALIGNMENT
- attribute LABEL for first macro parameter
- member argument for IF[N]DEF and .ERR[N]DEF directives
- integer initializer values for items of type [X]MMWORD
- optional name argument for .DATA, .DATA? and .CONST directives
- forward references in arguments for INVOKEd procedures
- overrides inside square brackets for base/index registers
- Optional Array Size for LABEL Directive

Some directives aren't touched by this option, although Masm won't accept them:

.X64 and .X64p INVOKE in 64-bit mode

runtime conditional directives .IF, .REPEAT, .WHILE in 64-bit mode

If these directives are to be disabled, it must be done with OPTION NOKEYWORD.

Option -zt: Set Name Decoration for STDCALL

Option -zt will fine-tune name decoration for STDCALL symbols. Syntax is:

-zt[0|1|2]

where value 0 will disable name decoration, value 1 will just add an underscore prefix and value 2 - which is the default - will emit full STDCALL name decoration as expected by most linkers.

Option -zt0 will make object modules compatible to ALINK + Win32.lib. It may also ease adding assembly modules to Borland's C++Builder or Delphi projects.

Option -Zv8: Enable Masm v8 Procedure Visibility

Option -Zv8 changes handling of procedure visibility to the way done by Masm v8+. See <u>Visibility of Procedures</u> for details.

Option -zze: Disable Export Name Decoration

Option -zze suppresses name decoration for procedures with the EXPORT attribute (exported name only).

Option -zzs: Avoid Wlink COFF Incompatibility

Option -zzs is kind of a workaround for a Wlink incompatibility. It's useful to be set if 1) the source module has a starting address, 2) output format is COFF AND 3) Wlink is to be used as linker.

3. Syntax Extensions

This chapter describes the syntax extensions of JWasm compared to Masm v8.

3.1 Directive INCBIN

This directive allows to include the contents of a file into the object module. Syntax is

INCBIN filename [, starting offset[, max size]]

<filename> should be enclosed in <> or double quotes.

3.2 FASTCALL Register Calling Convention

In **16- and 32-bit mode**, one may use either the Microsoft or the Watcom register calling convention. It's selected by option <u>-zf</u>.

The Microsoft FASTCALL convention uses registers AX, DX and BX in 16-bit for the first 3 parameters, and registers ECX and EDX in 32-bit for the first 2 parameters which are small enough to fit into a register.

The Open Watcom fastcall convention uses up to four registers (E/AX, E/DX, E/BX, E/CX).

In **64-bit mode**, FASTCALL means the standard Windows 64 ABI if output format is not ELF. For <u>-elf64</u>, there is no FASTCALL support implemented yet.

To make FASTCALL the default calling convention, there are 3 ways:

- Commandline options <u>-Gr</u> (or <u>-win64</u>)
- Second argument of the .MODEL directive: .MODEL FLAT, FASTCALL
- Directive OPTION LANGUAGE: FASTCALL

3.3 IDs enclosed in Back Quotes

IDs can be enclosed in back quotes (`) and thus they can contain characters not allowed in "normal" IDs. However, there is at least one case where IDs in back quotes won't be recognized: if the expansion operator (%) is located at position 0 in a line.

Example using back quotes:

Module 1:

```
`functionname.with.dots` PROC C PUBLIC a1:dword
```

Module 2:

```
`functionname.with.dots` PROTO C :dword .code
INVOKE `functionname.with.dots`, 1
```

Since IDs in back quotes are not 100% compatible with "normal" IDs, it might be considered to use the **ALIAS** directive instead. Be aware that, since the alias handling is a linker task, it is necessary to define both names, the alias name and the target name, as public.

3.4 Floating-Point Immediates in Instructions

Floating-point immediate values are accepted as instruction operands. As default, the type is a REAL4, which has a magnitude of 32 bits:

```
mov eax, 1.0
```

With type coercion, it's also possible to define a 64-bit "double", although it's probably useful in 64-bit code only:

```
mov rax, real8 ptr 1.0
```

Additionally, operators LOW32 and HIGH32 accept a floating-point constant as argument. In this case, the constant is assumed to have format REAL8. Thus it's possible to pass a double constant directly as a procedure argument in 32-bit code:

```
push HIGH32 1.0
push LOW32 1.0
call WorkWithReal8Value
```

3.5 Directive OPTION FIELDALIGN

OPTION FIELDALIGN sets the default value for structure alignment. The default value is 1 or the value set by cmdline switch -Zp. Syntax is:

OPTION FIELDALIGN: [1|2|4|8|16|32]

The current default value can be saved/restored with directives PUSHCONTEXT / POPCONTEXT_ALIGNMENT,

3.6 Directive OPTION PROCALIGN

With OPTION PROCALIGN parameter it's possible to automatically align procedures. Syntax is:

```
OPTION PROCALIGN: [1|2|4|8|16|32]
```

The default value is 1. The current value can be saved/restored with directives <u>PUSHCONTEXT / POPCONTEXT ALIGNMENT</u>,

Example:

```
.386
.model flat, stdcall
option PROCALIGN:16
.code

proc1 PROC
ret
proc1 endp

proc2 PROC
ret
proc2 endp

end
```

The listing shows that start address of *proc2* is aligned to 16 (=10h):

```
0000000
                                 proc1 PROC
0000000
                                     ret
          C3
0000000
                                 retn
0000001
                                 proc1 endp
0000010
                                 proc2 PROC
0000010
                                     ret
00000010
          C3
                                 retn
                                 proc2 endp
00000011
```

Note: to ensure that the procedures are aligned in the final binary as it is supposed by the OPTION PROCALIGN value, the alignment of the current code

segment must be at least the value of OPTION PROCALIGN.

3.7 Directive OPTION MZ

Directive OPTION MZ allows to fine-tune the values written to the MZ header if output format MZ (see <u>-mz</u> cmdline option) is selected. For other output formats, this option has no effect. The syntax for the directive is:

OPTION MZ:[start_fixups][:header_align][:heap_min][:heap_max]

The parameters are:

start_fixups offset within the header where segment fixups will start. The size

of the header will always be at least this value, even if there are no

fixups at all. Default - and minimum - value is 1Eh.

header_align alignment of the header (including segment fixups). Value must be

a power of 2, 10h is the default and minimum.

heap_min the additional space (in paragraphs) which is needed by the binary

to run. Default is the total of the sizes of the uninitialized BSS and

STACK segments.

heap_max space (in paragraphs) which the binary would like to have.

Default is FFFFh.

3.8 Directive OPTION ELF

Directive OPTION ELF allows to fine-tune the values written to the ELF header if output format ELF (see <u>-elf</u> or <u>-elf64</u>) is selected. For other output formats, this option has no effect. The syntax for the directive is:

OPTION ELF:osabi

The only argument <osabi> will be copied to the ELF header field EI_OSABI. It's a numeric constant, and according to the elf specs some valid values are:

0	ELFOSABI_NONE	unspecified
1	ELFOSABI_HPUX	HP-UX
2	ELFOSABI_NETBSD	NetBSD
3	ELFOSABI_LINUX	Linux, default
6	ELFOSABI_SOLARIS	Sun Solaris
7	ELFOSABI_AIX	IBM AIX
9	ELFOSABI_FREEBSD	FreeBSD
12	ELFOSABI_OPENBSD	OpenBSD

3.9 Directive OPTION WIN64

Directive OPTION WIN64 allows to set parameters for the Win64 output format if this format (see -win64 cmdline option) is selected. For other output formats, this option has no effect. The syntax for the directive is:

OPTION WIN64: switches

accepted values for switches are:

Store Register Arguments [bit 0]:

- 0: the "home locations" (also sometimes called "shadow space") of the first 4 register parameters are uninitialized. This is the default setting.
- 1: register contents of the PROC's first 4 parameters (RCX, RDX, R8 and R9) will be copied to the "home locations" within a PROC's prologue.

INVOKE Stack Space Reservation [bit 1]:

- 0: for each INVOKE the stack is adjusted to reserve space for the parameters required for the call. After the call, the space is released again. This is the default setting.
- 1: the maximum stack space required by all INVOKEs inside a procedure is computed by the assembler and reserved once on the procedure's entry. It's released when the procedure is exited. If INVOKEs are to be used outside of procedures, the stack space has to be reserved manually!

 Note: an assembly time variable, @ReservedStack, is created internally when this option is set. It will reflect the value computed by the assembler. It should also be mentioned that when this option is on, and a procedure contains no INVOKEs at all, then nevertheless the minimal amount of 4*8 bytes is reserved on the stack.

Warning: You should have understood exactly what this option does BEFORE you're using it. Using PUSH/POP instruction pairs to "save" values across an INVOKE is **VERBOTEN** if this option is on.

16-byte Alignment for Local Stack Variables [bit 2]:

0: standard 8-byte alignment for local variables.

1: 16-byte alignment for local variables. This setting is useful if you want to load or store XMM registers with instructions that expect aligned memory

references (i.e. MOVAPS). Note that variables with size $\!<\!16$ are not affected.

3.10 Directive OPTION FRAME

This option affects 64-bit only. It allows to make JWasm automatically generate prologues and epilogues for procedures with the FRAME attribute. Thus the code complies to the rules of <u>Win64 SEH</u> (Structured Exception Handling). Syntax is

OPTION FRAME:<AUTO | NOAUTO>

AUTO will enable this feature, NOAUTO (which is default) disables it.

The unwind information which is generated is "complete", that is, it contains the ".endprologue" pseudo-op already. To allow to save all non-volatile registers in the prologue, the "USES" phrase is more capable in this mode and will accept XMM registers to be saved and restored.

3.11 Directive OPTION RENAMEKEYWORD

This option allows to rename a keyword, so it can be used under a different name. Syntax:

OPTION RENAMEKEYWORD:<current_name>=new_name

current_name is the current name of the keyword and must be enclosed in angle brackets. *new_name* must be a valid identifier. If a keyword is to be renamed, it should be done at the beginning of the source, and a keyword shouldn't be renamed multiple times. Since v2.11, a keyword may be renamed temporarily and multiple times.

3.12 Directive OPTION DLLIMPORT

a) Using OPTION DLLIMPORT

This option makes the assembler assume that all PROTOs that follow this directive represent functions located in a dll. Syntax:

```
OPTION DLLIMPORT:<dll_name> | NONE
```

<dll_name> must be enclosed in angle brackets. Argument NONE will switch
back to the default mode.

b) Code Generation Effects

The effects of setting this options are subtle and useful only for MS Windows applications: if the function described by the prototype is called via INVOKE, slightly more efficient code than normal is generated, because the function's address in the IAT is used. Example:

```
INVOKE GetModuleHandle, NULL
```

code generation with OPTION DLLIMPORT:

```
push NULL
call DWORD PTR [_imp__GetModuleHandle@4]
```

code generation without OPTION DLLIMPORT:

```
push NULL
  call _GetModuleHandle@4
    ...
_GetModuleHandle@4:
    jmp DWORD PTR [_imp__GetModuleHandle@4] ;stub added by the
```

c) OPTION DLLIMPORT in Conjunction with -Fd Switch

Optionally, by using cmdline option <u>-Fd</u>, JWasm will write the import information received through OPTION DLLIMPORT lines to either a file or directly into the object module (COFF and ELF only). Example:

```
.386
.model flat,stdcall
option dllimport:<kernel32>
GetModuleHandleA proto :dword
ExitProcess proto :dword
option dllimport:none
.code
invoke GetModuleHandleA, 0
invoke ExitProcess, 0
end
```

JWasm -coff -**Fd=lnk.rsp** sample.asm

After the assembly step, file *lnk.rsp* will contain:

```
import '_ExitProcess@4' kernel32.ExitProcess
import ' GetModuleHandleA@4' kernel32.GetModuleHandleA
```

Both Open Watcom's Wlink and JWlink will be able to directly use this information and hence, as a result, no further Windows import libraries are needed in the link step:

Wlink format windows pe file sample.obj @lnk.rsp

JWlink may even go one step further - it's able to read import definitions contained in a COFF or ELF module's linker directive section (named ".drectve"). Therefore one can omit the filename argument for -Fd. Sample Win32_7 demonstrates the usage.

d) OPTION DLLIMPORT in Conjunction with -pe Switch

If output format PE is selected, using OPTION DLLIMPORT is the only way to resolve external references; see <u>PE Output Format</u> for more information.

3.13 Directive OPTION CODEVIEW

The OPTION CODEVIEW directive allows to fine-tune the generation of symbolic debugging information. It has no effect at all if commandline option <u>-</u> <u>Zi</u> isn't set. Syntax:

OPTION CODEVIEW: *switches*

There is currently only one switch:

[bit 0]: If 1, create symbols with indices $S_[L|G]$ THREAD32 instead of $S_[L|G]$ DATA32 for data items that are stored in segments with class 'TLS'. This allows the debugger to display the correct value of static TLS (Thread Local Storage) variables. Example:

```
option codeview:1

_TLS segment dword alias(".tls") public 'TLS'
tvar DD -1 ;codeview symbol tvar will be S_LTHREAD32
_TLS ends
```

3.13 Directive OPTION STACKBASE

The OPTION STACKBASE directive allows to change the way how stack variables - defined by the PROC and LOCAL directives - are accessed. Syntax is:

OPTION STACKBASE:register

register will be the index register that is used for accessing the stack variables. The "natural" register for accessing these variables is the [E|R]BP register (the "frame pointer"). With OPTION STACKBASE one might set any index register as frame pointer.

OPTION STACKBASE will additionally define assembly-time variable **@StackBase**. The assembler will add the value of this variable to the effective address of stack variables. @StackBase can be modified just like any userdefined variable - however, it is initialized to zero by the PROC directive. The purpose for the introduction of @StackBase is to make it feasible to use the "volatile" stack-pointer (ESP/RSP) register as base for accessing stack variables.

Finally, OPTION STACKBASE will define another assembly-time variable: **@ProcStatus**. This variable is read-only and will allow to query information about the current procedure's status. The information that is returned by **@ProcStatus** is:

Bit	Meaning if bit is 1
0	inside prologue of current procedure.
1	inside epilogue of current procedure.
2	"frame-pointer omission" is on for procedure
7	prologue of current procedure isn't created yet

3.15 Directives PUSHCONTEXT / POPCONTEXT ALIGNMENT

The PUSHCONTEXT / POPCONTEXT directives understand new qualifier ALIGNMENT, which saves/restores current values of <u>FIELDALIGN</u> and <u>PROCALIGN</u> options.

Note: in JWasm v2.00-2.09, PUSHCONTEXT / POPCONTEXT ALL did include these alignment values. Since JWasm v2.10, this is no longer true; this behavior is more Masm-compatible.

3.16 Directives .X64 and .X64p

These directives select a 64-bit (x86-64) cpu. In contrast to .X64, .X64p will allow to use privileged instructions.

The .X64 directive isn't needed usually, because for output formats WIN64 (see <u>win64</u>) and ELF64 (see <u>-elf64</u>), .X64 is the default.

The .X64p directive is useful for mixed-model binaries or system software (see example DOS64).

When the cpu is set to 64-bit, the SEGMENT directive accepts a new 'size' value: USE64. It tells the assembler that this segment's offset is 64-bit wide and uses 64-bit instructions.

The SYSCALL calling convention is renamed to SYSCALL_ when 64-bit is on, because there exists a SYSCALL instruction mnemonic in this mode.

3.17 Attribute LABEL for first Macro Parameter

The LABEL attribute for the first macro parameter allows access to a label which is assigned to the macro. Syntax is:

```
<macro_name> MACRO <param_name>:LABEL [,<param_name>[, ...]]
```

The LABEL attribute is accepted for the first parameter only. A macro with such a parameter can be invoked in the following way:

```
<label> <macro_name> [<argument>, ...]
```

Example:

```
foo macro lbl:LABEL, first, second lbl db first dw second endm
.data
data1 foo 1,1000
data2 foo 2,2000
```

Note that a code label (that is, a label followed by a colon or double-colon) is parsed BEFORE the macro is evaluated, hence such a label will have been defined already when the macro "runs".

3.18 Member Argument for IF[N]DEF and .ERR[N]DEF Directives

Since v2.07, JWasm's implementation of IF[N]DEF - and .ERR[N]DEF - will additionally accept a struct member as argument. This syntax requires a fully qualified name:

IFDEF <struct_name>.<member_name>

3.19 Initialization of Data Items with Type MMWORD/XMMWORD

For data items of types MMWORD or XMMWORD, JWasm will accept integer values for initialization:

```
vmm1 MMWORD 1122334455667788h
vxmm1 XMMWORD 112233445566778899AABBCCDDEEFFh
```

Masm will accept just floating-point initializers for data items of type [X]MMWORD. It's even worse, since floating-point initializers are **silently** ignored for data items with sizes != 4, 8 and 10; since XMMWORD has size 16, it's impossible to initialize such an item directly. JWasm copies this Masm behavior, but to allow to initialize a XMMWORD with a floating-point value, one may use type coercion:

```
vxmm1 XMMWORD real4 ptr 1.0 ;bytes 4-15 will be 0 vxmm2 XMMWORD real8 ptr 1.0 ;bytes 8-15 will be 0
```

Variants that work in both JWasm and Masm, and also allow to initialize the full XMMWORD are:

```
vxmm1 LABEL XMMWORD
real4 1.0, 2.0, 3.0, 4.0
vxmm2 LABEL XMMWORD
real8 1.0, 2.0
```

3.20 Optional Array Size for LABEL Directive

The LABEL directive accepts an optional array size argument:

```
myarray LABEL word : 10; myarray is assumed to be an array of 10 wo dw 1,2,3,4,5,6,7,8,9,10
```

Please note that the array size argument defines the array's number of elements, not the array's size in bytes.

This extension allows to define large initialized arrays, not restricted by line size limits. The SIZEOF and LENGTHOF operators will return the same values as if the array had been defined via the DUP operator.

The array size argument may be a forward reference:

```
myarray LABEL word : size_myarray / sizeof word
    dw 1,2,3,4,5,6,7,8,9,10
size_myarray equ $ - myarray
```

3.21 Optional Name Argument for Simplified Segment Directives

Masm allows an optional argument for simplified segment directives **.CODE**, **.FARDATA** and **.FARDATA?**. This is to set the name of the segment that is to be opened. JWasm will also accept the name argument for those directives; additionally, it's accepted for **.DATA**, **.DATA?** and **.CONST** directives.

3.22 Segment Attribute COMDAT

With segment attribute COMDAT one may define COMDAT sections. A COMDAT section is a section that can be defined by more than one object file. A COMDAT section must contain at least one symbol, the **COMDAT symbol**. The syntax to define the section is:

```
segname SEGMENT COMDAT( selection [, assoc_segment ] ) ...
```

The **selection** argument tells the linker what to do if multiple definitions of a COMDAT symbol are found; the accepted values are:

Value		Description
1		If the symbol is already defined, the linker issues a "multiply defined symbol" error.
2	anv	Any section that defines the same COMDAT symbol can be linked; the rest are removed.
3		The linker chooses an arbitrary section among the definitions for this symbol. If all definitions are not the same size, a "multiply defined symbol" error is issued.
4	exact	The linker chooses an arbitrary section among the definitions for this symbol. If all definitions do not match exactly, a "multiply defined symbol" error is issued.
5	associative	The section is linked if a certain other COMDAT section (see the assoc_segment in the syntax description) is linked.
6	largest	The linker chooses the largest definition from among all of the definitions for this symbol. If multiple definitions have this size, the choice between them is arbitrary.

Currently support for COMDAT is restricted to COFF.

JWasm won't do anything special with COMDAT sections; in future releases this may change: cmdline options similar to the Microsoft VC compiler options -Gf or -Gy may be added.

To create an object module that places each function in its own COMDAT

section (as it is done by MS VC if the -Gy option is given), it is recommended to use COMDAT in conjunction with ALIAS:

```
_TEXT_proc1 segment flat comdat(1) alias(".text")
proc1 proc c
...

proc1 endp
_TEXT1_proc1 ends

_TEXT_proc2 segment flat comdat(1) alias(".text")
proc2 proc c private
...

proc2 endp
_TEXT2_proc2 ends

_TEXT_proc3 segment flat comdat(1) alias(".text")
proc3 proc c
...

proc3 endp
_TEXT_proc3 ends
```

3.23 Attribute VARARGML for last Macro Parameter

VARARGML has mostly the same effects as attribute VARARG; the difference is that VARARGML will make the assembler concat lines if the last character on the line is a comma.

3.24 Miscellaneous

- 3.24.1 Numeric constant JWASM
- 3.24.2 Operating System Argument for .MODEL
- 3.24.3 Accepted Parameters for IF[N]DEF Directive
- 3.24.4 Visibility of Procedures
- 3.24.5 Non-RIP-Relative Addressing in 64-Bit

3.24.1 Numeric constant __JWASM__

__JWASM__ is a predefined symbol, its value is the current JWasm version * 100, that is, for v1.9 the value is 190. The predefined text equate @Version won't contain JWasm's version, for compatibility reasons it has value <800> (since v2.06, previously the value was <615>).

3.24.2 Operating System Argument for .MODEL

The .MODEL directive has an optional "operating system" argument. Masm accepts value OS_DOS only, JWasm accepts values OS_DOS and OS_OS2. This setting will affect the generated code of directives .STARTUP and .EXIT for 16-bit memory models.

3.24.3 Accepted Parameters for IF[N]DEF Directive

Masm's IF[N]DEF directive accepts user-defined symbols and registers, but fails for instructions, directives and other reserved words. JWasm's IF[N}DEF implementation accepts those symbols as well. OTOH, JWasm is a bit more picky and will display a warning if more than one item is found behind the directive - Masm just takes the first and silently skips the rest.

Also see Member Argument for IF[N]DEF and .ERR[N]DEF Directives.

3.24.4 Visibility of Procedures

When a PROTO or EXTERNDEF directive for a symbol is located in a module before a matching PROC directive, the visibility of this Procedure ("public" vs "private", or "external" vs. "static") is handled differently in Masm v6 or 7 and Masm v8 or newer:

Since Masm v8, a PROTO or EXTERNDEF for a symbol which is later defined as a PROC will make the procedure public, no matter what a possible visibility attribute of the PROC itself - or the default one set with OPTION PROC - is telling.

OTOH, with Masm v6/7, both the visibility attribute of the PROC directive and the current default setting of OPTION PROC will affect the symbol's visibility.

	Masm6	Masm8	JWasm	JWasm+Zv8
On,E,P		X		X
On, E, Pn		X		X
On, E, Pp	Χ	X	X	X
0p,E,P	X	X	Х	X
Op,E,Pn	Χ	X		X
Op,E,Pp	Χ	X	X	X

On = OPTION PROC:PRIVATE
Op = OPTION PROC:PUBLIC

E = PROTO or EXTERNDEF before PROC
P = PROC without visibility attribute

Pn = PROC with PRIVATE visibility attribute Pp = PROC with PUBLIC visibility attribute

x = procedure will be public

As default, JWasm more or less copies the Masm v6/7 behavior. The difference is that an explicite visibility attribute behind PROC has the highest priority for JWasm. However, since v2.04, there's an additional cmdline option -Zv8 which will make JWasm behave like Masm v8+.

It should be noted that without a PROTO/EXTERNDEF before PROC, there are no differences between Masm v6, v8 and JWasm, and the -Zv8 switch also has no effect then.

3.24.5 Non-RIP-Relative Addressing in 64-Bit

In 64-bit a RIP-relative addressing mode was introduced. This mode is used as default for direct addressing, because it allows to access all code or data labels with a 32-bit displacement. However, for constant address locations this mode is not appropriate and won't be used:

```
mov al, gs:[1000h]
mov al, gs:[100000000h]
```

If a constant address is to be accessed without segment prefixes - note that in 64-bit, only segment registers FS and GS can be used as segment prefixes - one has to use the FLAT keyword in JWasm:

```
mov al, FLAT:[1000h] ;invalid for Masm mov al, FLAT:[100000000h] ;invalid for Masm
```

This syntax differs from Masm's, because Masm won't accept FLAT. In 64-bit Masm, the syntax for accessing constant addresses is instead:

```
mov al, [1000h] ;invalid for JWasm mov al, [100000000h] ;invalid for JWasm
```

The code that will be generated won't show any differences:

4. Instruction Sets

JWasm supports all instructions sets supported by Masm v8. These are

- the instructions implemented by 8086, 80186, 80286, 80386, 80486.
- the Pentium and Pentium Pro instructions.
- the MMX and K3D instruction set extensions.
- the SSE, SSE2, SSE3 and SSSE3 instruction set extensions.
- the Intel VMX instruction set extension (since JWasm v2.09).
- the x86-64 64-bit instruction set (implemented by ML64).

Additionally supported are

- SSE4.1 and SSE4.2 instruction sets (since JWasm v2.01).
- AVX instruction set (since JWasm v2.06).

With Masm, SSE4.1 and SSE4.2 require Masm v9; AVX requires Masm v10.

5. Code Generation Differences

JWasm might generate slightly different code than Masm on some occasions. Commandline option <u>-Zg</u> should eliminate most of these differences. However, some differences are due to fixed Masm bugs (see below), in which case option -Zg won't have any effect.

For a few instructions, the encoding differs between Masm versions.

Example:

```
cmp al,dl
```

is encoded 38 D0 in Masm v6, but 3A C2 in Masm v8. In such cases, JWasm will prefer to copy the encoding of Masm v8.

5.1 Forward References in Macro Expressions

Like Masm, JWasm usually evaluates expressions in preprocessor directives during the first pass only. However, due to different jump optimization strategies of Masm and JWasm, the results may differ. This is very unlikely to impose a problem, but it is mentioned here for completeness. An example (found in README.TXT of Masm v6.14):

```
Label1:
    JMP Label2
Label2:

REPEAT Label2 - Label1
    INC AX
ENDM
```

Masm will - incorrectly - repeat the loop 10 times, although the result of expression Label 2 - Label 1 is 2 only. OTOH, JWasm will repeat the loop 2 times only, because it's using an "optimistic" strategy concerning forward references.

6. Output Formats

The format of the assembler's output is selected by commandline options. These are:

<u>-omf</u>	Intel's "relocatable Object Module Format", including 32-bit MS extensions. It's the default.
-coff	(MS) COFF object module format, supports flat memory models only. It's usually used for 32-bit Windows modules.
<u>-elf</u>	32-Bit ELF ("Executable and Linkable Format") object modules.
<u>-elf64</u>	64-bit ELF object modules.
<u>=</u> win64	64-bit format for Win64 object modules. COFF variant with 64-bit extensions.
-djgpp	COFF variant used by DJGPP. Not active in precompiled binaries.
<u>-bin</u>	raw binary format.
<u>-mz</u>	DOS MZ binary format.
<u>-pe</u>	32- and 64-bit PE binaries.

The formats selected by -bin, -mz and -pe are **binary** formats, hence the output is not supposed to be fed to a linker.

6.1 OMF Output Format

The OMF format fully supports the segmented architecture of the x86 cpu. The specification has been extended around 1990 to support 32-bit, including the FLAT memory model. However, since it's a bit old now, there's no support of some of the new relocations introduced for 64-bit.

- OMF won't support the IMAGEREL and SECTIONREL operators.
- JWasm currently won't emit OMF LIDATA records (unlike Masm). This means that if the assembly source contains lots of arrays defined with DUP, the object module might become significantly larger than a similar object module written by Masm.
- JWasm emits a few COMENT records that Masm won't. These are remnants of Open Watcom's WASM; they may be suppressed with commandline options <u>-zlc</u> and <u>-zld</u>. They shouldn't do any harm, though.

The OMF object module consists of records of certain types. JWasm writes those records in the following order:

Type		Description
Type		
THEADR	1	Marks the start of an object module, contains the name of the source file.
		MS extension; tells linker the CodeView version number. Emitted if -Zi option is active.
COMENT, class E9h	n	Borland-style auto-dependency record(s). Emitted if -Zd or -Zi options are active.
COMENT, class 9Eh	1	MS extension; directs the linker to use a "standardized" segment ordering. Emitted if the .DOSSEG directive is found in the source.
COMENT, class 9Fh	n	Default library search name(s). Emitted if INCLUDELIB directives are used.
LNAMES	n	Defines names for segments and groups (optionally also for externals, communals and publics).
SEGDEF	n	Defines segment(s).
COMENT, class FEh	n	Tells WLink to do FARCALL optimization for a specific segment. Suppressed with -zld.
GRPDEF	n	Defines group(s).

EXTDEF	n Defines external(s).
COMENT, class A8h	Defines weak external(s). Emitted if EXTERN directive is used with "altname".
COMDEF	n Defines communal(s).
ALIAS	n Defines alias(es).
PUBDEF	n Defines public(s).
COMENT, class A0h	Defines export(s). Emitted if the EXPORT attribute is used in a PROC directive.
COMENT, class A2h	Link Pass Separator. Emitted if no entry point is defined in this object module.
LEDATA	n Data record; defines segment content(s).
FIXUPP	Contains information to make the linker resolve references between object modules.
	n Line number information. Emitted if -Zd or -Zi option is active.
COMENT, class FDh	Tells disassembler that a code segment contains data. Suppressed with -zlc.
MODEND	Marks the end of the object module; optionally defines the entry point.
TEDATA I	EIVLIDD and I INNI IM records usually are intermixed; a FIVLIDD

LEDATA, FIXUPP and LINNUM records usually are intermixed; a FIXUPP and/or a LINNUM record appears immediately after the LEDATA record to which it refers.

Since v2.11, multiple THEADR records are written if cmdline options -Zd or -Zi are set and line number information is written for more than one source file; this conforms to MASM behavior.

6.2 COFF Output Format

The COFF object module format is suitable for flat, non-segmented memory models. In most cases the COFF format is used for 32-bit code. However, it's possible to create a 64-bit COFF module if the current cpu is xx64 when the .MODEL directive is parsed (note that -win64 is the usual way to create modules for 64-bit Windows). To some extent 16-bit code is also supported with COFF, although OMF will always be the better choice then.

- The -safeseh cmdline option and the .SAFESEH directive are supported by COFF only.
- Not supported by COFF are:
 - the **GROUP** directive
 - the directives to control segment order: .ALPHA, .DOSSEG, .SEQ
 - code that will generate "far" fixups (i.e. direct FAR calls or jumps).
- A few, common segment names are translated in the output module when COFF is selected:
 - name TEXT is translated to .text
 - name _DATA is translated to .data
 - o name CONST is translated to .rdata
 - name BSS is translated to .bss

This also affects simplified segment directives, since _TEXT, _DATA, CONST and _BSS segments are internally generated when directives .CODE, .DATA, .CONST or .DATA? are detected.

- If COFF is selected, Masm will ignore segment combine type STACK and treat stack segments just like normal data segments (the MS linker and compatibles will set the stack's size with commandline options). JWasm DOES NOT ignore combine type STACK for the COFF output format; additionally, unless the segment contains initialized data, **JWasm will reset the size of stack segments to 0**.
- Directive **EXTERN name (altname) : <type>** will define a weak external with characteristics IMAGE_WEAK_EXTERN_SEARCH_LIBRARY (see MS PE and COFF specification).
- Directive ALIAS <aliasname>=<actual name> will define a weak external with characteristics
 IMAGE WEAK EXTERN SEARCH ALIAS (see MS PE and COFF

- specification).
 There are some cmdline options to control the extent of symbols that are written to the COFF output module: -zlf, -zlf and -zlf.

6.3 Win64 Output Format

6.3.1 Win64 Basics

This output format, that is selected with the <u>-win64</u> commandline option, is a variant of the <u>COFF output format</u>. It's commonly used to create object modules for 64-bit Windows. The default calling convention is the Win64 <u>FASTCALL</u> implementation.

6.3.2 Directive INVOKE in Win64

Unlike the 64-bit version of Masm, which doesn't support INVOKE anymore, JWasm still does; however, please be aware of some peculiarities:

- in theory, using INVOKE requires the FRAME attribute for PROC. It will work without FRAME, but Win64 SEH won't be happy with it then.
- the default implementation in 64-bit is very simple: for each INVOKE register RSP is reduced by the space required for the arguments, then the call is issued and finally register RSP is restored. To enable a more efficient code generation see OPTION WIN64, INVOKE Stack Space Reservation this option is not active as default because it virtually requires a certain programming style.
- there is no additional check that the stack is aligned to 16 byte. The PROC's FRAME attribute ensures that the stack is correctly aligned after the prologue is done. However, it's the programmers responsibility that the stack is still aligned when the code generated by INVOKE starts.
- parameter names listed behind the PROC directive will always refer to the
 parameter's shadow space on the stack. However, on a procedure's entry the
 actual values of the first four parameters are hold in registers, and the value
 of the associated shadow spaces are undefined. See OPTION WIN64, Store
 Register Arguments, how to make JWasm automatically save the register
 arguments and thus initialize the shadow space on a procedure's entry.

6.3.3 Win64 Structured Exception Handling (SEH)

SEH in Win64 differs significantly from the implementation in Win32. It's very

well possible to ignore Win64 SEH for assembly. However, if an assembly routine wants to comply to these rules, a thorough understanding of the Win64 ABI is necessary. Masm (the 64-bit version) supplies some "primitives" for SEH support (.ALLOCSTACK, .PUSHREG, .SAVEREG, ...), along with a new FRAME attribute for the PROC directive. These features are also supported by JWasm. See sample Win64 3 how the "primitives" are to be used for SEH support.

The big disadvantage is that using the FRAME keyword in Masm "disables" most of the other high level features combined with PROC (function parameters, locals and registers saved with USES) because no function prologues and epilogues are generated anymore. Additionally, the implementation at least in Masm v8 seems to be a bit buggy, at least in Masm v8. Because of this and to ease the usage of SEH in Win64 there is a new directive implemented in JWasm:

OPTION FRAME: AUTO

If this option is set, JWasm will create Win64 SEH-compatible prologues and epilogues. If the option is off, JWasm will behave Masm-compatible, that is, FRAME found in a PROC directive will disable automatic prologue/epilogue generation. See sample Win64 3e how this option is supposed to be used.

As for the PROC syntax: The Masm documentation states that FRAME can be used in combination with USES and procedure parameters and must be located behind all parameters. However, this syntax isn't accepted by any Masm version. The only syntax which Masm will accept without being confused is FRAME as the one and only parameter for PROC. Therefore JWasm doesn't follow the Masm documentation in this point: the optional FRAME keyword is expected *before* the procedure parameters. The syntax in JWasm is:

```
procname PROC [public] FRAME[:exc_handler] [USES <reglist>] [par
```

The SEH "primitives" will generate some additional data in segments .pdata and .xdata. This data is somewhat hidden, but JWasm will display the corresponding data definitions in the listing if option -Sg is set.

Finally, JWasm's default behavior of INVOKE isn't fully SEH-compatible, because the stack pointer is temporarily changed to make room for arguments.

To make INVOKE comply to SEH, <u>OPTION WIN64</u> *INVOKE Stack Space Reservation* has to be used.

6.4 Binary Output Format

If the binary output format has been selected, the contents of the file are just the raw data bytes emitted by the assembler, no header, relocations or symbol tables are generated. All references have to be resolved internally.

The binary format is most useful for bootloaders or DOS COM files, but may be used to create any binary format. See sample Win32 5, that demonstrates how the binary format is used to create a Win32 application.

If a listing file is produced, a binary map will be added, which shows the file and memory layout of the image:

```
.model tiny
                                     .data
0000000
         0D0A48656C6C6F2C20
                                         db 13,10,"Hello, world!",13,
                                str1
0000000
                                     .code
                                     org 100h
                                start:
00000100
00000100
         B409
                                     mov ah, 09h
                                    mov dx, offset str1
00000102
         BA0000
00000105 CD21
                                     int 21h
                                     mov ax, 4c00h
00000107
          B8004C
0000010A
         CD21
                                     int 21h
                                     end start
```

Binary Map:

Segment	Pos(file)	VA	Size(fil)	Size(mem)
_TEXT	0	100	С	С
_DATA	C	10C	12	12

Note that bytes with "undefined contents" at the start and the end of the output file are skipped and won't become part of the binary.

6.5 PE Output Format

The (Windows) PE output format (both 32- and 64-bit) is a binary format - there's no link step supposed to follow the assembly step. Hence all references must be resolved internally, no external references are possible. Since the Windows ABI is implemented as a set of dlls that export function names, it's necessary to provide a mechanism to call such external functions - in the PE format this is achieved with the help of directive OPTION DLLIMPORT; it allows to attach a module name to function prototypes used by the assembly source and consequently give the assembler the means to resolve all references without the help of a linker.

The PE format requires a .MODEL FLAT directive in the source code. This directive will trigger the creation of the internal PE-header section, the value of the cpu at this time will determine whether a 32- or 64-bit PE binary is to be written.

As in all binary formats, the listing will contain a binary map; see <u>Binary Output</u> Format for more details.

If -pe is set, a few sections will be created internally:

- -.hdr\$1 : section will contain the DOS MZ-stub.
- -.hdr\$2: section will contain the PE header
- -.hdr\$3 : section will contain the PE object table

The default values in the PE header are

Field	32-bit Value	64-bit Value (if different)
Signature	"PE"	
Machine	14Ch	8664h
Timestamp	date & time	
Size OptionalHeader	0E0h	0F0h
Characteristics	10Fh	12Fh
Magic	10Bh	20Bh
LinkerVersion	5.1	

ImageBase	400000h	
SectionAlignment	1000h	
FileAlignment	200h	
OSVersion	4.0	
ImageVersion	0.0	
SubsystemVersion	4.0	
Win32Version	0	
Checksum	0	
Subsystem	2 (=Console)	
DllCharacteristics	0	
SizeOfStack	100000h,1000h	
SizeOfHeap	100000h,1000h	
LoaderFlags	0	

To change the default values in the PE header there are two options. First, a predefined assembly-time variable **@pe_file_flags** will map the value of field *Characteristics* - changing the value of **@pe_file_flags** will also change the value in the header field. The other fields in the PE header are only accessible by setting the appropriate values in section .hdr\$2. Fields not listed in the table above are set internally by the assembler to ensure data integrity and cannot be modified from within the assembly source.

If the PE binary is to use **resources**, a .rsrc section has to be created which is to contain them. Defining the resources manually works and is doable, but it might become tedious if a lot of resource items are to be defined. Therefore tool **res2inc** is supplied, which allows to convert a compiled resource file (.RES) to an assembly include file.

If a **dll** is created with -pe, one has to mark all procedures that are to be exported with the EXPORT attribute. If the exported names are to be undecorated, use the -zze cmdline option.

Sample <u>Win64_8</u> shows how a 64-bit Windows binary is created with -pe. It also shows how to define resources in .rsrc manually and how to modify default values of PE header fields.

7. Masm bugs fixed in JWasm

#	Description	Fixed Masm Version
1	the infamous "invoke" bug: if an argument for invoke has to be expanded (from BYTE or WORD to DWORD, for example), bad code was generated.	9
2	PROTOs contained twice in the source caused an EXTDEF entry to be generated in the object module.	-
3	"TYPE xmm0" will return 10 in Masm v6 and v7 - the correct value is 16.	8
4	a nested structure might cause a GPF in Masm if the embedded STRUCT's starting offset has to be adjusted due to alignment.	-
5	defining huge arrays in Masm is very slow and might even cause a deadlock if COFF has been selected as output format.	-
6	for Masm v6 and v7, if an array > 64 kB is defined and output format OMF is selected, the array's size will be mod $0x10000$ only.	8
7	Masm doesn't flag invalid numbers in struct/array initializer strings.	-
8	if an ALIAS is defined somewhere in the source and the symbol table is listed, a 'General Failure' error occurs in Masm if output format is OMF.	-
9	Type "coerces" for DWORD data items defined in a 32bit segment are ignored by Masm, i.e., "dd far16 ptr <symbol>" will generate a near32 fixup instead of a far16 one.</symbol>	-
	if the ALIGN directive has to add 5 bytes in 32bit code segments, Masm includes an "add eax,0" opcode, which isn't a no-op because flags are modified.	-
11	silent truncation of immediate constants: Masm v6 and v7 will accept line "mov [word_variable],12345h" without error.	8
12	preprocessed output with option -EP may erroneously contain text macros and macro function calls if the macros are located in the initialization string of a structured variable.	-
	Masm generates wrong code if a conditional jump is coupled with a	

13	type coercion which modifies offset magnitude. Examples: "jz near32 ptr" in 16bit code or "jz near16 ptr" in 32bit code).	-
14	if the arguments given to Masm end with an option which expects a parameter (i.e. "ml -c -Fo"), a 'General Failure' may occur.	-
15	floating-point data items in Masm can be followed by any suffix (example: REAL4 1.0foo, 2.0bar). JWasm won't accept this.	-
16	If a local is defined inside a macro, Masm will create a unique name for it. The name is constructed by using '??' as prefix, followed by a hexadecimal number with 4 digits. There is no check for overflow, however, so if the total of locals in all macros exceeds 65536, strange errors will occur.	-
17	If a weak external is defined for -coff with the ALIAS directive, an invalid fixup - and also a strange entry in the module's symbol table - is created.	-
18	If a section contains more than 0xffff relocations in COFF, the number of relocations that is stored in the object module is just the value of the lower 16-bit half of the relocation count.	8
19	If a symbolic constant (=equate) is made public in OMF format, Masm will store the symbol's value in a 16-bit record if it is in the range -32768 65535. If the symbol is referenced in another module as a 32-bit number, it is always zero-extended, never sign-extended; hence values -132768 will become 65535 32768.	-
20	if data labels become public by the -Zf option (and not by the PUBLIC directive), their names are not decorated. Also, if format is COFF, they won't become true publics, they're just included in the symbol table with class "static",	-

It's slightly dangerous to fix old Masm bugs, since some code might work only if the bugs exists. So no, JWasm won't achieve 100% Masm compatibility.

8. Optional Features

There exist some features that are not activated in the precompiled binaries. To enable these options, the JWasm source must be recompiled with one or more switches set in the command line:

Support for Djgpp's variant of COFF

This option is enabled with switch **-DDJGPP_SUPPORT**. The generated JWasm binary will accept an additional output format (-djgpp) to be set in the commandline, that makes the assembler generate COFF modules compatible with DJGPP.

Support for Assembly Source generated by Intel C++ compiler

This option is enabled with switch **-DDOTNAMEX**. It will make the assembler accept names that contain dots if the name starts with a dot or an underscore. Note that this behavior is not compatible with standard Masm syntax.

9. Known Bugs and missing Features

a) Bugs which are known but not fixed yet:

There are currently no known bugs.

b) Features which aren't implemented yet:

- directives PAGE, TITLE, SUBTITLE, SUBTTL. the directives are ignored and a warning (level 4) is displayed.
- the following parameters of the OPTION directive:
 - OLDMACROS
 - EXPR16
 - READONLY
- optional parameter NONUNIQUE for structures is ignored.
- commandline option -AT (Enable tiny model).
- commandline options -Sp, -Ss and -St (set page length, subtitle and title).
- commandline option -H (set max external name length).
- commandline option -Ta (Assemble non-.ASM file).
- commandline option -Zd for ELF output format.
- commandline option -Zi for ELF output format.

If there's ever a problem with one of the missing features, it's very likely that it's related to OPTION OLDMACROS. This option makes Masm 6 emulate two Masm 5.1 peculiarities:

- macro arguments may be separated by spaces
- ampersands (&) inside the macro must match the current level of macro nesting.

c) Missing features which most likely won't be implemented:

- syntax "mm(n)" and "xmm(n)" (supported by Masm v6 and v7 only)
- commandline option -Bl, -F, -Fe, -Fm and -link: since JWasm doesn't launch a linker, those options are useless.
- commandline options -Fr and -FR (generate browser info).
- commandline option -Sc (generate timings in listing).
- commandline option -errorReport (Report internal assembler errors to Microsoft).

10. License

This manual was written by Andreas Grech (aka Japheth).

It may be redistributed as long as it is free of charge.

Appendix A. JWasm Reserved Words

Reserved Words are case-insensitive. Besides the items listed below all instruction mnemonics are also Reserved Words.

Registers 16- and 32-bit Modes

8-bit registers	AL	CL	DL	BL	AH	CH	DH	BH
16-bit registers	AX	CX	DX	BX	SP	BP	SI	DI
32-bit registers	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
Segment registers	ES	CS	SS	DS	FS	GS		
Floating-point registers	ST	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
MMX registers	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
SSE registers	XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7
AVX registers	YMM0	YMM1	YMM2	YMM3	YMM4	YMM5	YMM6	YMM7
Control registers	CR0		CR2	CR3	CR4			
Debug registers	DR0	DR1	DR2	DR3			DR6	DR7
Test registers ^[1]				TR3	TR4	TR5	TR6	TR7

[1]: invalid in 64-bit mode.

Additional Registers in 64-bit Mode

8-bit registers					SPL	BPL	SIL	DIL
	R8B	R9B	R10B	R11B	R12B	R13B	R14B	R15B
16-bit registers	R8W	R9W	R10W	R11W	R12W	R13W	R14W	R15W
32-bit registers	R8D	R9D	R10D	R11D	R12D	R13D	R14D	R15D
64-bit registers	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
	R8	R9	R10	R11	R12	R13	R14	R15
SSE registers	XMM8	XMM9	XMM10	XMM11	XMM12	XMM13	XMM14	XMM15
AVX registers	YMM8	YMM9	YMM10	YMM11	YMM12	YMM13	YMM14	YMM15
Control registers	CR8							

Types

BYTE

SBYTE

WORD

SWORD

DWORD

SDWORD

REAL4

FWORD

QWORD

SQWORD

REAL8

TBYTE

REAL10

OWORD

YMMWORD

NEAR

FAR

NEAR16

NEAR32

FAR16

FAR32

MMWORD

XMMWORD

Unary Operators

.TYPE
HIGH
HIGH32
HIGHWORD
IMAGEREL ^[1]
LENGTH
LENGTHOF
LOW
LOW32
LOWWORD
LROFFSET
MASK
OFFSET
OPATTR
SECTIONREL ^[1]
SEG
SHORT
SIZE
SIZEOF
THIS
TYPE
WIDTH

[1]: not for OMF output format.

Binary Operators

EQ

NE

GE

GT

LE

LT

MOD

PTR

DUP

SHL^[1]

SHR^[1]

AND^[1]

OR^[1]

XOR^[1]

[1]: these keywords are also instructions.

Directives

.8086 .186 .286 .286C .286P .386 .386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM .X64
.286 .286C .286P .386 .386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.286C .286P .386 .386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.286P .386 .386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.386 .386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.386C .386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.386P .486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.486 .486P .586 .586P .686 .686P .K3D .MMX .XMM
.486P .586 .586P .686 .686P .K3D .MMX .XMM
.586 .586P .686 .686P .K3D .MMX .XMM
.586P .686 .686P .K3D .MMX .XMM
.686 .686P .K3D .MMX .XMM
.686P .K3D .MMX .XMM
.K3D .MMX .XMM .X64
.MMX .XMM .X64
.XMM .X64
.X64
.X64P
.8087
.287
.387
.NO87
.CREF
.LIST
.LISTALL
.LISTIF, .LFCOND
.NOCREF, .XCREF
.NOLIST, .XLIST
.NOLISTIF, .SFCOND

.TFCOND
PAGE
SUBTITLE, SUBTTL
TITLE
.LISTMACRO, .XALL
.LISTMACROALL, .LALL
.NOLISTMACRO, .SALL
.ALPHA
.DOSSEG, DOSSEG
.SEQ
.CODE
.STACK
.DATA
.DATA?
.FARDATA
.FARDATA?
.CONST
.IF
.REPEAT
.WHILE
.BREAK
.CONTINUE
.ELSE
.ELSEIF
.ENDIF
.ENDW
.UNTIL
.UNTILCXZ
.EXIT
.STARTUP
.MODEL
.RADIX
.SAFESEH

.ERR
.ERR1
.ERR2
.ERRE
.ERRNZ
.ERRDIF
.ERRDIFI
.ERRIDN
.ERRIDNI
.ERRB
.ERRNB
.ERRDEF
.ERRNDEF
COMMENT
IF
IFE
IF1
IF2
IFDIF
IFDIFI
IFIDN
IFIDNI
IFB
IFNB
IFDEF
IFNDEF
ELSE
ELSEIF
ELSEIFE
ELSEIF1
ELSEIF2
ELSEIFDIF
ELSEIFDIFI

ELSEIFIDN
ELSEIFIDNI
ELSEIFB
ELSEIFNB
ELSEIFDEF
ELSEIFNDEF
ENDIF
FOR, IRP
FORC, IRPC
REPEAT, REPT
WHILE
MACRO
EXITM
ENDM
GOTO
PURGE
INCLUDE
TEXTEQU
CATSTR
SUBSTR
INSTR
SIZESTR
DB
DW
DD
DF
DQ
DT
STRUCT, STRUC
UNION
TYPEDEF
RECORD
COMM

EXTERN, EXTRN
EXTERNDEF
PUBLIC
PROTO
PROC
ENDP
LOCAL
LABEL
INVOKE
ORG
ALIGN
EVEN
SEGMENT
ENDS
GROUP
ASSUME
ALIAS
ECHO, %OUT
END
EQU
INCBIN
INCLUDELIB
NAME
OPTION
POPCONTEXT
PUSHCONTEXT

Additional Directives in 64-bit Mode

.ALLOCSTACK

.ENDPROLOG

.PUSHFRAME

.PUSHREG

.SAVEREG

.SAVEXMM128

.SETFRAME

Other Reserved Words

ADDR

FLAT

VARARG

FRAME^[1]

C

SYSCALL^[2]

STDCALL

PASCAL

FORTRAN

BASIC

FASTCALL

[1]: in 64-bit mode only.

[2]: in 64-bit, calling convention SYSCALL is renamed to SYSCALL_, since in this mode there exists a SYSCALL instruction.

Appendix B. Source Samples

Win64 3 - SEH Support in Win64

Win64_3e - SEH Support in Win64 (JWasm specific)

DOS64 - Switch to Long Mode and Back

Win32_5 - Create a Win32 Binary with -bin

Win32_7 - Usage of OPTION DLLIMPORT and -Fd Switch

Win64_8 - Create a Win64 Binary with -pe

Win64_3 - SEH Support in Win64

```
;--- This sample shows how to use SEH primitives. It doesn't use hll
;--- directives. Thus this source can be assembled by both JWasm
;--- and Masm64.
;--- to assemble enter:
;--- JWasm -win64 Win64 3.asm
;--- or:
;--- ml64 -c Win64 3.asm
;--- to link the binary enter:
;--- Link Win64_3.obj
   option casemap:none
   includelib kernel32.lib
   includelib user32.lib
HINSTANCE typedef QWORD
         typedef QWORD
HWND
         typedef QWORD
HMENU
        typedef QWORD
HICON
        typedef QWORD
HBRUSH
HCURSOR typedef QWORD
WPARAM typedef QWORD
LPARAM
        typedef QWORD
         typedef QWORD
LPSTR
LPVOID
         typedef QWORD
         typedef DWORD
UINT
NULL
              egu 0
WS_OVERLAPPEDWINDOW equ OCF0000h
CW_USEDEFAULT equ 80000000h
SW_SHOWDEFAULT equ 10
SW SHOWNORMAL equ 1
IDC ARROW
              equ 32512
IDI_APPLICATION equ 32512
WM_DESTROY equ 2
CS_VREDRAW
              equ 1
              equ 2
CS_HREDRAW
COLOR WINDOW equ 5
```

```
proto_WNDPROC typedef proto :HWND,:QWORD,:WPARAM,:LPARAM
WNDPROC typedef ptr proto_WNDPROC
WNDCLASSEXA struct 8
                        ?
cbSize
                DWORD
style
                DWORD
                        ?
lpfnWndProc
                WNDPROC ?
cbClsExtra
                DWORD
cbWndExtra
                DWORD
hInstance
                HINSTANCE ?
hIcon
                HICON
                        ?
hCursor
                HCURSOR ?
hbrBackground
                HBRUSH
                        ?
lpszMenuName
                LPSTR
lpszClassName
                LPSTR
                        ?
hIconSm
                HICON
                        ?
WNDCLASSEXA ends
POINT
        struct
    SDWORD
Χ
    SDWORD ?
У
POINT
        ends
MSG struct 8
hwnd
        HWND
message DWORD
                2
wParam WPARAM
                ?
1Param
                ?
        LPARAM
time
        DWORD
                ?
рt
        POINT
                <>
MSG ends
GetModuleHandleA proto :LPSTR
GetCommandLineA proto
ExitProcess
                 proto :UINT
LoadIconA
                 proto :HINSTANCE, :LPSTR
LoadCursorA
                 proto :HINSTANCE, :LPSTR
RegisterClassExA proto :ptr WNDCLASSEXA
                 proto :DWORD, :LPSTR, :LPSTR, :DWORD, :SDWORD, :SDW
CreateWindowExA
ShowWindow
                 proto :HWND, :SDWORD
UpdateWindow
                 proto :HWND
GetMessageA
                 proto :ptr MSG, :HWND, :SDWORD, :SDWORD
```

proto :HWND, :UINT, :WPARAM, :LPARAM

TranslateMessage **proto** :ptr MSG DispatchMessageA **proto** :ptr MSG PostQuitMessage **proto** :SDWORD

DefWindowProcA

```
;WinMain proto :HINSTANCE, :HINSTANCE, :LPSTR, :UINT
    .data
ClassName db "SimpleWinClass", 0
AppName db "Our First Window", 0
    .data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
    .code
WinMainCRTStartup proc FRAME
    push
           rbp
    .pushreq rbp
    mov
           rbp,rsp
    .setframe rbp, 0
    .endprolog
           rsp, 32
    sub
           ecx, NULL
    mov
    call
           GetModuleHandleA
    mov
           hInstance, rax
    call
           GetCommandLineA
    mov
           CommandLine, rax
           rcx, hInstance
    mov
           rdx, NULL
    mov
    mov
           r8, CommandLine
           r9d, SW_SHOWDEFAULT
    mov
    call
           WinMain
    mov
           ecx, eax
    call
           ExitProcess
    align 4
WinMainCRTStartup endp
WinMain proc FRAME
    push rbp
    .pushreg rbp
          rbp, rsp
    mov
    .setframe rbp, 0
    .endprolog
          rsp, sizeof WNDCLASSEXA + sizeof MSG + sizeof HWND + 12*8
    sub
hInst
          equ <[rbp+10h]>
```

```
hPrevInst equ <[rbp+18h]>
          equ <[rbp+20h]>
CmdLine
CmdShow
          equ <[rbp+28h]>
     equ <[rbp - sizeof WNDCLASSEXA].WNDCLASSEXA>
WC
    equ <[rbp - sizeof WNDCLASSEXA - sizeof MSG].MSG>
msq
hwnd equ <[rbp - sizeof WNDCLASSEXA - sizeof MSG - sizeof HWND]>
   mov
          hInst, rcx ;store param1 in shadow space
          wc.cbSize, SIZEOF WNDCLASSEXA
   mov
   mov
         wc.style, CS HREDRAW or CS VREDRAW
         rax, OFFSET WndProc ; using LEA is preferable
   mov
    lea
         rax, [WndProc]
   mov
         wc.lpfnWndProc, rax
   mov
         wc.cbClsExtra, NULL
         wc.cbWndExtra, NULL
   mov
   mov
         wc.hInstance, rcx
         wc.hbrBackground, COLOR_WINDOW+1
   mov
   mov
         wc.lpszMenuName, NULL
         rax, OFFSET ClassName ;using LEA is preferable
    mov
    lea
          rax, [ClassName]
   mov
         wc.lpszClassName, rax
   mov
          ecx, NULL
          edx, IDI_APPLICATION
   mov
    call LoadIconA
   mov
         wc.hIcon, rax
         wc.hIconSm, rax
   mov
   mov
          ecx, NULL
          edx, IDC_ARROW
   mov
    call LoadCursorA
    mov
         wc.hCursor,rax
          rcx, wc
    lea
    call RegisterClassExA
          ecx, NULL
    mov
          rdx, [ClassName]
    lea
    lea
          r8, [AppName]
   mov
          r9d, WS OVERLAPPEDWINDOW
          dword ptr [rsp+4*8], CW_USEDEFAULT
   mov
          dword ptr [rsp+5*8], CW_USEDEFAULT
   mov
   mov
          dword ptr [rsp+6*8], CW_USEDEFAULT
          dword ptr [rsp+7*8], CW_USEDEFAULT
   mov
          qword ptr [rsp+8*8], NULL
   mov
          gword ptr [rsp+9*8], NULL
   mov
          rax, hInst
   mov
          [rsp+10*8], rax
   mov
          qword ptr [rsp+11*8], NULL
   mov
```

```
call CreateWindowExA
    mov hwnd, rax
    mov rcx, hwnd
    mov edx, SW_SHOWNORMAL
    call ShowWindow
    mov rcx, hwnd
    call UpdateWindow
;--- message loop
@@:
        lea rcx, msg
        mov rdx, NULL
        mov r8, 0
        mov r9, 0
        call GetMessageA
        and rax, rax
        jz @F
        lea rcx, msg
        call TranslateMessage
        lea rcx, msg
        call DispatchMessageA
        jmp @B
@@:
          rax, msg.wParam
    mov
    add
          rsp, sizeof WNDCLASSEXA + sizeof MSG + sizeof HWND + 12*8
          rbp
    pop
    ret
    align 4
WinMain endp
WndProc proc FRAME
          rsp, 4*8
    sub
    .allocstack 4*8
    .endprolog
    cmp edx, WM_DESTROY
    jnz @F
    mov ecx, NULL
    call PostQuitMessage
    xor rax, rax
    jmp exit
@@:
    call DefWindowProcA
exit:
    add rsp, 4*8
    ret
    align 4
```

WndProc **endp**

end

Win64_3e - SEH Support in Win64 (JWasm specific)

```
;--- SEH support in Win64. Unlike Win64_3,
;--- this version uses hll directives, so it cannot be assembled
;--- with Masm64. Also, OPTION FRAME: AUTO is used.
;--- to create the binary enter:
;--- JWasm -win64 Win64 3e.asm
;--- Link Win64 3e.obj
    option casemap:none
                        ; generate SEH-compatible prologues and epil
    option frame:auto
    includelib kernel32.lib
    includelib user32.lib
HINSTANCE typedef QWORD
HWND
         typedef QWORD
        typedef QWORD
HMENU
        typedef QWORD
HICON
HBRUSH typedef QWORD
HCURSOR typedef QWORD
WPARAM typedef QWORD
         typedef QWORD
LPARAM
         typedef QWORD
LPSTR
         typedef QWORD
LPVOID
UINT
         typedef DWORD
NULL
              equ 0
WS_OVERLAPPEDWINDOW equ OCF0000h
CW_USEDEFAULT equ 80000000h
SW_SHOWDEFAULT equ 10
SW_SHOWNORMAL equ 1
IDC ARROW
              equ 32512
IDI_APPLICATION equ 32512
WM_DESTROY equ 2
CS VREDRAW
              equ 1
CS_HREDRAW
              equ 2
COLOR_WINDOW
              equ 5
```

```
proto_WNDPROC typedef proto :HWND,:QWORD,:WPARAM,:LPARAM
WNDPROC typedef ptr proto_WNDPROC
WNDCLASSEXA struct 8
                        ?
cbSize
                DWORD
style
                DWORD
                        ?
lpfnWndProc
                WNDPROC ?
cbClsExtra
                DWORD
cbWndExtra
                DWORD
hInstance
                HINSTANCE ?
hIcon
                HICON
                        ?
hCursor
                HCURSOR ?
hbrBackground
                HBRUSH
                        ?
lpszMenuName
                LPSTR
lpszClassName
                LPSTR
                        ?
hIconSm
                HICON
                        ?
WNDCLASSEXA ends
POINT
        struct
    SDWORD
Χ
    SDWORD ?
У
POINT
        ends
MSG struct 8
hwnd
        HWND
message DWORD
                2
wParam WPARAM
                ?
1Param
                ?
        LPARAM
time
        DWORD
                ?
рt
        POINT
                <>
MSG ends
GetModuleHandleA proto :LPSTR
GetCommandLineA proto
ExitProcess
                 proto :UINT
LoadIconA
                 proto :HINSTANCE, :LPSTR
LoadCursorA
                 proto :HINSTANCE, :LPSTR
RegisterClassExA proto :ptr WNDCLASSEXA
                 proto :DWORD, :LPSTR, :LPSTR, :DWORD, :SDWORD, :SDW
CreateWindowExA
ShowWindow
                 proto :HWND, :SDWORD
UpdateWindow
                 proto :HWND
GetMessageA
                 proto :ptr MSG, :HWND, :SDWORD, :SDWORD
```

proto :HWND, :UINT, :WPARAM, :LPARAM

TranslateMessage **proto** :ptr MSG DispatchMessageA **proto** :ptr MSG PostQuitMessage **proto** :SDWORD

DefWindowProcA

```
WinMain proto : HINSTANCE, : HINSTANCE, : LPSTR, : UINT
    . data
ClassName db "SimpleWinClass", 0
AppName db "Our First Window", 0
    .data?
hInstance HINSTANCE ?
CommandLine LPSTR ?
    .code
WinMainCRTStartup proc FRAME
    invoke GetModuleHandleA, NULL
           hInstance, rax
    mov
    invoke GetCommandLineA
           CommandLine, rax
   mov
    invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess, eax
WinMainCRTStartup endp
WinMain proc FRAME hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPS
    local wc:WNDCLASSEXA
    local msq:MSG
    local hwnd:HWND
   mov
          hInst, rcx
         wc.cbSize, SIZEOF WNDCLASSEXA
   mov
         wc.style, CS_HREDRAW or CS_VREDRAW
   mov
    lea
        rax, [WndProc]
         wc.lpfnWndProc, rax
   mov
         wc.cbClsExtra, NULL
   mov
         wc.cbWndExtra, NULL
   mov
         wc.hInstance, rcx
   mov
         wc.hbrBackground, COLOR_WINDOW+1
   mov
   mov
         wc.lpszMenuName, NULL
        rax, [ClassName]
    lea
   mov
         wc.lpszClassName, rax
    invoke LoadIconA, NULL, IDI_APPLICATION
         wc.hIcon, rax
   mov
          wc.hIconSm, rax
   mov
    invoke LoadCursorA, NULL, IDC_ARROW
```

```
mov
         wc.hCursor,rax
    invoke RegisterClassExA, addr wc
    invoke CreateWindowExA, NULL, ADDR ClassName, ADDR AppName,\
           WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, \
           CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, \
           hInst, NULL
          hwnd, rax
    mov
    invoke ShowWindow, hwnd, SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .while (1)
        invoke GetMessageA, ADDR msg, NULL, 0, 0
        .break .if (!rax)
        invoke TranslateMessage, ADDR msg
        invoke DispatchMessageA, ADDR msg
    .endw
         rax, msg.wParam
    mov
    ret
WinMain endp
WndProc proc FRAME hWnd: HWND, uMsg: UINT, wParam: WPARAM, 1Param: LPARA
    .if ( edx == WM_DESTROY )
        invoke PostQuitMessage, NULL
        xor rax, rax
    .else
        invoke DefWindowProcA, rcx, edx, r8, r9
    .endif
    ret
WndProc endp
end WinMainCRTStartup
```

DOS64 - Switch to Long Mode and Back

```
;--- DOS program which switches to long-mode and back.
;--- Note: requires at least JWasm v2.
;--- Also: needs a 64bit cpu in real-mode to run.
;--- Parts of the source are based on samples supplied by
;--- sinsi and Tomasz Grysztar in the FASM forum.
;--- To create the binary enter:
:--- JWasm -mz DOS64.asm
    .x64p
;--- 16bit start/exit code
_TEXT16 segment use16 para public 'CODE'
    assume ds:_TEXT16
    assume es:_TEXT16
                     ; Global Descriptors Table Register
GDTR label fword
                        ; limit of GDT (size minus one)
    dw 4*8-1
                      ; linear address of GDT
    dd offset GDT
                      ; Interrupt Descriptor Table Register
IDTR label fword
                        ; limit of IDT (size minus one)
   dw 256*16-1
    dd ⊙
                        ; linear address of IDT
nullidt label fword
    dw 3FFh
    0 bb
    align 8
                           ; null descriptor
GDT dq 0
    dw OFFFFh, 0, 9A00h, 0AFh ; 64-bit code descriptor
    dw OFFFFh, 0, 9A00h, 000h ; compatibility mode code descriptor
    dw OFFFFh,0,9200h,000h ; compatibility mode data descriptor
wPICMask dw 0 ; variable to save/restore PIC masks
start16:
    push cs
    pop ds
```

```
mov ax, cs
    movzx eax, ax
    shl eax, 4
    add dword ptr [GDTR+2], eax ; convert offset to linear address
    mov word ptr [GDT+2*8+2], ax
   mov word ptr [GDT+3*8+2], ax
    shr eax, 16
   mov byte ptr [GDT+2*8+4], al
   mov byte ptr [GDT+3*8+4], al
   mov ax, ss
   mov dx, es
    sub ax, dx
   mov bx, sp
    shr bx,4
    add bx, ax
    mov ah, 4Ah
    int 21h
                  ; free unused memory
    push cs
    pop es
   mov ax, ss
    mov dx, cs
    sub ax, dx
    shl ax,4
    add ax, sp
    push ds
    pop ss
    mov sp,ax ; make a TINY model, CS=SS=DS=ES
    smsw ax
    test al,1
    jz @F
    mov dx, offset err1
    mov ah, 9
    int 21h
   mov ah, 4Ch
    int 21h
err1 db "Mode is V86. Need REAL mode to switch to LONG mode!", 13, 10,
@@:
    xor edx, edx
    mov eax,80000001h ; test if long-mode is supported
    cpuid
    test edx, 20000000h
    jnz @F
   mov dx, offset err2
    mov ah, 9
    int 21h
```

```
mov ah, 4Ch
   int 21h
err2 db "No 64bit cpu detected.",13,10,'$'
   mov bx, 1000h
   mov ah, 48h
   int 21h
   jnc @F
   mov dx, offset err3
   mov ah, 9
   int 21h
   mov ah, 4Ch
   int 21h
err3 db "Out of memory", 13, 10, '$'
@@:
   add ax, 100h-1; align to page boundary
   mov al, 0
   mov es, ax
;--- setup page directories and tables
   sub di, di
   mov cx, 4096
   sub eax, eax
   sub di, di
   mov ax, es
   movzx eax, ax
   shl eax, 4
   lea edx, [eax+5000h]
   mov dword ptr [IDTR+2], edx
   or eax, 111b
   add eax, 1000h
   mov es:[di+0000h], eax ; first PDP table
   add eax, 1000h
   mov es:[di+1000h],eax ; first page directory
   add eax, 1000h
   mov es:[di+2000h], eax ; first page table
                       ; address of first page table
   mov di, 3000h
   mov eax, 0 + 111b
   mov cx, 256
                         ; number of pages to map (1 MB)
@@:
   stosd
```

```
add di,4
    add eax, 1000h
    loop @B
;--- setup ebx/rbx with linear address of _TEXT
    mov bx,_TEXT
    movzx ebx, bx
    shl ebx,4
    add [llg], ebx
;--- create IDT
    mov di,5000h
    mov cx,32
    mov edx, offset exception
    add edx, ebx
make_exc_gates:
    mov eax, edx
    stosw
    mov ax,8
    stosw
    mov ax, 8E00h
    stosd
    xor eax, eax
    stosd
    stosd
    add edx, 4
    loop make_exc_gates
    mov cx, 256-32
make_int_gates:
    mov eax, offset interrupt
    add eax, ebx
    stosw
    mov ax,8
    stosw
    mov ax, 8E00h
    stosd
    xor eax, eax
    stosd
    stosd
    loop make_int_gates
    mov di,5000h
    mov eax, ebx
    add eax, offset clock
    mov es:[di+80h*16+0],ax ; set IRQ 0 handler
```

```
shr eax, 16
   mov es:[di+80h*16+6],ax
   mov eax, ebx
    add eax, offset keyboard
   mov es:[di+81h*16+0],ax ; set IRQ 1 handler
    shr eax, 16
   mov es:[di+81h*16+6],ax
;--- clear NT flag
    pushf
    pop ax
    and ah, OBFh
    push ax
    popf
;--- reprogram PIC: change IRQ 0-7 to INT 80h-87h, IRQ 8-15 to INT 8
    cli
    in al, 0A1h
   mov ah, al
    in al,21h
   mov [wPICMask],ax
   mov al, 10001b ; begin PIC 1 initialization
   out 20h, al
   mov al, 10001b ; begin PIC 2 initialization
    out 0A0h, al
                    ; IRQ 0-7: interrupts 80h-87h
   mov al,80h
   out 21h, al
                   ; IRQ 8-15: interrupts 88h-8Fh
   mov al,88h
    out 0A1h, al
   mov al, 100b
                     ; slave connected to IRQ2
    out 21h, al
   mov al, 2
   out 0A1h, al
                      ; Intel environment, manual EOI
   mov al,1
    out 21h, al
    out 0A1h, al
    in al, 21h
   mov al, 11111100b ; enable only clock and keyboard IRQ
    out 21h, al
    in al, 0A1h
   mov al, 11111111b
    out 0A1h, al
   mov eax, cr4
```

```
or eax,1 shl 5
            ; enable physical-address extensions (PAE)
  mov cr4,eax
  mov ecx, 0C0000080h ; EFER MSR
  rdmsr
  or eax,1 shl 8 ; enable long mode
  wrmsr
  lgdt [GDTR]
  lidt [IDTR]
  MOV CX, SS
  shl ecx, 4
  movzx esp, sp
  mov eax, cr0
  or eax,8000001h
  db 66h, 0EAh ; jmp 0008:0000000
llg dd offset long_start
  dw 8
;--- switch back to real-mode and exit
backtoreal:
  cli
  mov eax, cr0
  and eax, 7FFFFFFFh ; disable paging
  mov cr0, eax
  mov ecx,0C0000080h ; EFER MSR
  rdmsr
  wrmsr
             ; set SS,DS and ES to 64k data
  mov ax, 24
  mov ss, ax
  mov ds, ax
  mov es, ax
             ; switch to real mode
  mov eax,cr0
  and al, OFEh
  mov cr0, eax
```

```
db 0eah
                      ; clear instruction cache, CS=real-mode seg
   dw $+4
   dw _TEXT16
   mov ax, STACK
                      ; SS=real-mode seg
   mov ss, ax
   mov sp, 4096
   push cs
                      ; DS=real-mode _TEXT16 seg
   pop ds
   lidt [nullidt] ; IDTR=real-mode compatible values
   mov eax, cr4
   and al, not 20h ; disable physical-address extensions (PAE)
   mov cr4, eax
;--- reprogram PIC: change IRQ 0-7 to INT 08h-0Fh, IRQ 8-15 to INT 7
   mov al, 10001b
                  ; begin PIC 1 initialization
   out 20h, al
   mov al, 10001b ; begin PIC 2 initialization
   out 0A0h, al
   mov al,08h
                     ; IRQ 0-7: back to ints 8h-Fh
   out 21h, al
   mov al,70h
                     ; IRQ 8-15: back to ints 70h-77h
   out 0A1h,al
                   ; slave connected to IRQ2
   mov al, 100b
   out 21h, al
   mov al, 2
   out 0A1h, al
   mov al, 1
                      ; Intel environment, manual EOI
   out 21h, al
   out 0A1h, al
   in al,21h
   mov ax,[wPICMask] ; restore PIC masks
   out 21h, al
   mov al, ah
   out 0A1h, al
   sti
   mov ax, 4c00h
   int 21h
_TEXT16 ends
```

```
;--- here's the 64bit code segment.
;--- since 64bit code is always flat but the DOS mz format is segmen
;--- there are restrictions, because the assembler doesn't know the
;--- linear address where the 64bit segment will be loaded:
;--- + direct addressing with constants isn't possible (mov [0B8000h
;--- since the rip-relative address will be calculated wrong.
;--- + 64bit offsets (mov rax, offset <var>) must be adjusted by the
;--- address where the 64bit segment was loaded (is in rbx).
;--- rbx must preserve linear address of _TEXT
_TEXT segment para use64 public 'CODE'
    assume ds:FLAT, es:FLAT
long_start:
    xor eax, eax
    mov ss, eax
    mov esp, ecx
    sti
                    ; now interrupts can be used
    call WriteStrX
    db "Hello 64bit", 10,0
nextcmd:
                ; r8b will be filled by the keyboard irg routine
   mov r8b,0
nocmd:
    cmp r8b, 0
    iz nocmd
                   ; ESC?
    cmp r8b, 1
    jz esc_pressed
                    ; 'r'?
    cmp r8b, 13h
    jz r_pressed
    call WriteStrX
    db "unknown key ",0
    mov al, r8b
    call WriteB
    call WriteStrX
    db 10,0
    imp nextcmd
;--- 'r' key: display some register contents
r_pressed:
    call WriteStrX
    db 10, "cr0=", 0
    mov rax, cr0
```

```
call WriteQW
    call WriteStrX
    db 10, "cr2=", 0
    mov rax, cr2
    call WriteOW
    call WriteStrX
    db 10, "cr3=", 0
    mov rax, cr3
    call WriteOW
    call WriteStrX
    db 10, "cr4=", 0
    mov rax, cr4
    call WriteQW
    call WriteStrX
    db 10, "cr8=", 0
    mov rax, cr8
    call WriteQW
    call WriteStrX
    db 10,0
    jmp nextcmd
;--- ESC: back to real-mode
esc_pressed:
    jmp [bv]
by label fword
    dd offset backtoreal
    dw 16
;--- screen output helpers
;--- scroll screen up one line
;--- rsi = linear address start of last line
;--- rbp = linear address of BIOS area (0x400)
scroll_screen:
    cld
    mov edi,esi
    movzx eax,word ptr [rbp+4Ah]
    push rax
    lea rsi, [rsi+2*rax]
    mov cl, [rbp+84h]
    mul cl
    mov ecx, eax
    rep movsw
    pop rcx
    mov ax, 0720h
    rep stosw
```

```
ret
WriteChr:
    push rbp
    push rdi
    push rsi
    push rbx
    push rcx
    push rdx
    push rax
    mov edi, 0B8000h
    mov ebp, 400h
    cmp byte ptr [rbp+63h], 0B4h
    jnz @F
    xor di, di
@@:
    movzx ebx, word ptr [rbp+4Eh]
    add edi, ebx
    movzx ebx, byte ptr [rbp+62h]
    mov esi, edi
    movzx ecx, byte ptr [rbx*2+rbp+50h+1] ;ROW
    movzx eax, word ptr [rbp+4Ah]
    mul ecx
    movzx edx, byte ptr [rbx*2+rbp+50h] ;COL
    add eax, edx
    mov dh,cl
    lea edi, [rdi+rax*2]
    mov al, [rsp]
    cmp al, 10
    jz newline
    mov [rdi], al
    mov byte ptr [rdi+1], 07
    inc dl
    cmp dl, byte ptr [rbp+4Ah]
    jb @F
newline:
    mov d1, 00
    inc dh
    cmp dh, byte ptr [rbp+84h]
    ibe @F
    dec dh
    call scroll_screen
@@:
    mov [rbx*2+rbp+50h], dx
    pop rax
    pop rdx
    pop rcx
```

```
pop rbx
    pop rsi
    pop rdi
    pop rbp
    ret
WriteStr: ;write string in rdx
    push rsi
    mov rsi, rdx
    cld
@@:
    lodsb
    and al, al
    jz @F
    call WriteChr
    jmp @B
@@:
    pop rsi
    ret
WriteStrX: ;write string at rip
    push rsi
    mov rsi, [rsp+8]
    cld
@@:
    lodsb
    and al, al
    jz @F
    call WriteChr
    jmp @B
@@:
    mov [rsp+8],rsi
    pop rsi
    ret
WriteQW: ;write QWord in rax
    push rax
    shr rax, 32
    call WriteDW
    pop rax
WriteDW:
    push rax
    shr rax, 16
    call WriteW
    pop rax
WriteW:
    push rax
```

```
shr rax,8
    call WriteB
    pop rax
         ;write Byte in al
WriteB:
    push rax
    shr rax, 4
    call WriteNb
    pop rax
WriteNb:
    and al, OFh
    add al, '0'
    cmp al, '9'
    jbe @F
    add al,7
@@:
    jmp WriteChr
;--- exception handler
exception:
excno = 0
    repeat 32
    push excno
    jmp @F
    excno = excno+1
    endm
@@:
    call WriteStrX
    db 10,"Exception ",0
    pop rax
    call WriteB
    call WriteStrX
    db " errcode=",0
    mov rax,[rsp+0]
    call WriteQW
    call WriteStrX
    db " rip=",0
    mov rax,[rsp+8]
    call WriteQW
    call WriteStrX
    db 10,0
@@:
    jmp $
;--- clock and keyboard interrupts
clock:
```

```
push rbp
   mov ebp, 400h
    inc dword ptr [rbp+6Ch]
    pop rbp
                  ; handler for all other interrupts
interrupt:
    push rax
   mov al, 20h
    out 20h, al
    pop rax
    iretq
keyboard:
    push rax
    in al, 60h
    test al,80h
    jnz @F
   mov r8b, al
@@:
                ; give finishing information
; to keyboard...
    in al,61h
    out 61h, al
   mov al,20h
    out 20h, al
                 ; ...and interrupt controller
    pop rax
    iretq
_TEXT ends
;--- 4k stack, used in both modes
STACK segment use16 para stack 'STACK'
    db 4096 dup (?)
STACK ends
    end start16
```

Win32_5 - Create a Win32 Binary with -bin

```
;--- Win32 "hello world" console application.
;--- Uses JWasm's bin output format, so no linker needed.
;--- assemble: JWasm -bin -Fo Win32 5.exe Win32 5.ASM
    .386
   option casemap:none
    .nolist
    include winnt.inc ;include PE image definitions
    .list
STD_OUTPUT_HANDLE equ -11
IMAGEBASE equ 400000h
PEHDR segment dword FLAT
;--- define the DOS "MZ" header
   org IMAGEBASE
    IMAGE_DOS_HEADER <"ZM", 80h, 1, 0,4,0,-1,0,200h,0,0,0,0,0,<0>,0,
   db 0Eh
             ;push cs
    db 1Fh
                  ;pop ds
   db OBAh, OEh, O ; mov dx, text
    db 0B4h,09h ; mov ah,9
   db 0CDh, 21h ;int 21h
    db 0B8h, 01h, 4Ch; mov ax, 4c01h
    db 0CDh, 21h ;int 21h
   db "This program cannot be run in DOS mode",13,10,'$'
   org IMAGEBASE+80h
;--- define the Win32 "PE" header
PEHdr label byte
   db "PE", 0, 0
```

```
IMAGE_FILE_RELOCS_STRIPPED or IMAGE_FILE_EXECUTABLE_IMAGE or
    IMAGE_OPTIONAL_HEADER32 { 10Bh, ;magic
                                        ;linker major, minor
         6,0,
        1000h, 1000h, 0,
                                        ; sizeof code, initialized data,
         IMAGEREL mainCRTStartup, ;entry point
        IMAGEREL start_text, IMAGEREL start_rdata, ;baseof code, da
        IMAGEBASE, ;imagebase
        1000h, 200h, ;section alignment, file alignment
                        ;OS major, minor
        4,0,
                       ;Image major, minor
        0,0,
        4,0,
                       ;Subsys major, minor
                       ;win32 version
         Θ,
                    ;sizeof image
        3000h,
                       ;sizeof header
        1000h,
                        ;checksum
         IMAGE SUBSYSTEM WINDOWS CUI,
                        ;dll characteristics
        100000h, 1000h, ; stack res, com
        100000h, 1000h, ; heap res, com
        Θ,
                       ;loader flags
        16,
                        ;number of directories
        <<0,0>,
                   ;exports
        < IMAGEREL start_idata, SECTIONREL endof_idata >, ;imports
        <0,0>,<0,0>,
    ;resource, exception
<>>,<>,<>,<>,
    ;security, baserelocs, debug, architecture
<>>,<>,<>,<>,
    ;globalptr, tls, load_config, bound_import
<>>,<>,<>>,<>>;iat, delay_import, com descriptor, reserve
;--- define the section table
sectiontable label byte
    IMAGE_SECTION_HEADER <".text", <sizeof_text>, IMAGEREL start_tex
         200h, 0, 0, 0, 0, 060000020h >
    IMAGE SECTION HEADER <".rdata", <SECTIONREL endof idata + sizeof
         400h, 0, 0, 0, 0, 040000040h >
num_sections equ ( $ - sectiontable ) / sizeof IMAGE_SECTION_HEADER
    org IMAGEBASE+200h ; forces physical size of header to 200h and
PEHDR ends
;--- the ALIGNx segments are needed because
;--- section alignment and file alignment are different
ALIGN1 segment dword public FLAT 'DATA'
```

IMAGE FILE HEADER < IMAGE FILE MACHINE I386, num sections, 0, 0,

```
org OEOOh ; change pc to RVA 1000h
ALIGN1 ends
_TEXT segment dword public FLAT 'CODE'
TEXT ends
ALIGN2 segment dword public FLAT 'DATA'
    org OEOOh ; change pc to RVA 2000h
ALIGN2 ends
_IDATA segment dword public FLAT 'DATA'
start_rdata label byte
start_idata label byte
;--- import descriptors go here
IDATA ends
_IDATA$1 segment dword public FLAT 'DATA'
    IMAGE_IMPORT_DESCRIPTOR <<0>,0,0,0,0
;--- ILT entries go here
_IDATA$1 ends
_IDATA$2 segment dword public FLAT 'DATA'
           ;--- end of last ILT
;--- IAT entries go here
_IDATA$2 ends
_IDATA$3 segment dword public FLAT 'DATA'
    dd 0 ;--- end of last IAT
;--- import name strings go here
_IDATA$3 ends
_IDATA$4 segment dword public FLAT 'DATA'
endof_idata equ $
_IDATA$4 ends
CONST segment dword public FLAT 'DATA'
start_const label byte
CONST ends
DefineImpDll macro name
_IDATA segment
    IMAGE_IMPORT_DESCRIPTOR <<IMAGEREL name&ILT>,0,0,IMAGEREL name,
_IDATA ends
IDATA$1 segment
ifdef ImportDefined
    dd 0 ;terminate previous ILT
endif
name&ILT label dword
_IDATA$1 ends
IDATA$2 segment
ifdef ImportDefined
```

```
dd 0 ;terminate previous IAT
endif
name&IAT label dword
IDATA$2 ends
_IDATA$3 segment
name db @CatStr(!",name, !"),0
    align 4
IDATA$3 ends
ImportDefined equ 1
    endm
DefineImport macro name
_IDATA$1 segment
    dd IMAGEREL n&name
_IDATA$1 ends
_IDATA$2 segment
lp&name typedef ptr pr&name
        lp&name IMAGEREL n&name
name
_IDATA$2 ends
_IDATA$3 segment
n&name dw 0
    db @CatStr(!",name, !"),0
    align 4
IDATA$3 ends
    endm
prWriteConsoleA typedef proto stdcall :dword, :dword, :dword, :dword
prGetStdHandle typedef proto stdcall :dword
prExitProcess typedef proto stdcall :dword
    DefineImpDll kernel32
    DefineImport ExitProcess
    DefineImport WriteConsoleA
    DefineImport GetStdHandle
if 0 ;if further dlls are to be imported
prMessageBoxA typedef proto stdcall :dword, :dword, :dword, :dword
    DefineImpDll user32
    DefineImport MessageBoxA
endif
CONST segment
string db 13,10, "hello, world.",13,10
sizeof_const equ $ - start_const
```

```
CONST ends
_TEXT segment
    assume ds:FLAT, es:FLAT
start_text label near
;--- start of program
main proc
local dwWritten:dword
local hConsole:dword
    invoke GetStdHandle, STD_OUTPUT_HANDLE
    mov
            hConsole, eax
    invoke WriteConsoleA, hConsole, addr string, sizeof string, add
    xor
            eax, eax
    ret
main endp
;--- entry
mainCRTStartup proc c
    invoke main
    invoke ExitProcess, eax
mainCRTStartup endp
sizeof_text equ $ - start_text
    org 200h   ;align size of _TEXT to next 512 byte boundary
_TEXT ends
    end
```

Win32_7 - Usage of OPTION DLLIMPORT and -Fd Switch

```
;--- Win32 7 - Shows how to use OPTION DLLIMPORT and switch -Fd.
               No import libraries are needed in the link step.
;--- assemble: JWasm -coff -Fd Win32_7.ASM
;--- link: JWlink format windows pe f Win32_7.0BJ
    .386
    .model FLAT, stdcall
    option casemap:none
STD_OUTPUT_HANDLE equ -11
   option dllimport:<kernel32>
WriteConsoleA proto :dword, :dword, :dword, :dword, :dword
GetStdHandle proto :dword
ExitProcess
             proto :dword
   option dllimport:<user32>
MessageBoxA proto :dword, :dword, :dword
  option dllimport:<none>
    .const
msq db 13,10, "hello, world.",13,10
    db 0
    .code
main proc
local
       written:dword
    invoke GetStdHandle, STD_OUTPUT_HANDLE
    mov ebx, eax
    invoke WriteConsoleA, ebx, addr msg, sizeof msg,
                addr written, 0
    invoke MessageBoxA, 0, addr msg, 0, 0
    ret
```

```
main endp
;--- entry
start:
    invoke main
    invoke ExitProcess, 0
    end start
```

Win64_8 - Create a Win64 Binary with -pe

```
;--- create a 64-bit binary with -pe cmdline option
      JWasm -pe Win64_8.asm
; - - -
                       ; -pe requires to set cpu, model & language
    .x64
    .model flat, fastcall
   option casemap:none
   option frame:auto ; generate SEH-compatible prologues and epil
                       ; init shadow space, reserve stack at PROC 1
   option win64:3
;--- resource IDs
IDR MENU1 equ 100
IDR_BITMAP1 equ 101
IDM_EXIT equ 1000
NULL
         equ 0
         typedef ptr
LPSTR
LPVOID
         typedef ptr
UINT
         typedef dword
B00L
         typedef dword
;--- winbase definitions
HINSTANCE typedef ptr
;--- winuser definitions
HWND typedef ptr
         typedef ptr
HMENU
         typedef ptr
HICON
HBRUSH typedef ptr
HCURSOR typedef ptr
         typedef ptr
HDC
HBITMAP
         typedef ptr
         typedef ptr
WPARAM
         typedef qword
LPARAM
WS_OVERLAPPEDWINDOW equ OCF0000h
```

CW_USEDEFAULT equ 80000000h

```
SW SHOWDEFAULT equ 10
SW_SHOWNORMAL
               equ 1
IDC ARROW
               equ 32512
IDI_APPLICATION equ 32512
CS VREDRAW
               egu 1
CS_HREDRAW
               equ 2
               equ 5
COLOR WINDOW
WM DESTROY
               equ 2
WM_PAINT
               equ 000Fh
WM COMMAND
               equ 0111h
proto_WNDPROC typedef proto :HWND,:qword,:WPARAM,:LPARAM
WNDPROC typedef ptr proto_WNDPROC
WNDCLASSEXA struct 8
cbSize
                dword
                         ?
                dword
                         ?
style
lpfnWndProc
                WNDPROC ?
cbClsExtra
                dword
                         ?
cbWndExtra
                dword
                         ?
hInstance
                HINSTANCE ?
hIcon
                HICON
                HCURSOR ?
hCursor
hbrBackground
                HBRUSH
lpszMenuName
                LPSTR
                         ?
lpszClassName
                LPSTR
                         ?
hIconSm
                HICON
                         ?
WNDCLASSEXA ends
POINT
        struct
    sdword ?
Χ
    sdword ?
POINT
        ends
MSG struct 8
                ?
hwnd
        HWND
                ?
message dword
wParam WPARAM
1Param
        LPARAM
                ?
                ?
time
        dword
pt
        POINT
                <>
MSG ends
RECT struct
left
        sdword
                ?
        sdword
top
```

```
riaht
        sdword ?
bottom sdword ?
RECT ends
PAINTSTRUCT struct 8
hdc
           HDC ?
fErase
           B00L ?
rcPaint
           RECT <>
fRestore
           B00L ?
fIncUpdate BOOL ?
rgbReserved byte 32 dup (?)
PAINTSTRUCT ends
;--- wingdi definitions
DIB RGB COLORS equ 0
SRCCOPY
                equ 00CC0020h
HGDIOBJ typedef ptr
BITMAPINFOHEADER struct
biSize
                dword
                        ?
biWidth
                sdword
biHeiaht
                sdword
                        ?
biPlanes
                word
biBitCount
                word
                        ?
                        ?
biCompression
                dword
                        ?
biSizeImage
                dword
biXPelsPerMeter sdword
                       ?
biYPelsPerMeter sdword
biClrUsed
                dword
biClrImportant
                dword
BITMAPINFOHEADER ends
    option dllimport:<kernel32>
GetModuleHandleA proto :LPSTR
GetCommandLineA proto
ExitProcess
                 proto :UINT
    option dllimport:<user32>
BeginPaint
                 proto :HWND, :ptr PAINTSTRUCT
CreateWindowExA
                 proto :dword, :LPSTR, :LPSTR, :dword, :sdword, :sdw
DefWindowProcA
                 proto :HWND, :UINT, :WPARAM, :LPARAM
DestroyWindow
                 proto :HWND
DispatchMessageA proto :ptr MSG
EndPaint
                 proto :HWND, :ptr PAINTSTRUCT
GetClientRect
                 proto :HWND, :ptr RECT
```

```
GetMessageA
                proto :ptr MSG, :HWND, :sdword, :sdword
                proto :HINSTANCE, :LPSTR
LoadBitmapA
                proto :HINSTANCE, :LPSTR
LoadCursorA
LoadIconA
                proto :HINSTANCE, :LPSTR
PostQuitMessage proto :sdword
RegisterClassExA proto :ptr WNDCLASSEXA
ShowWindow
               proto :HWND, :sdword
TranslateMessage proto :ptr MSG
UpdateWindow proto:HWND
    option DLLIMPORT:<qdi32>
BitBlt
                  proto :HDC, :dword, :dword, :dword, :HDC,
CreateCompatibleDC proto :HDC
DeleteDC
                  proto :HDC
GetDIBits
                  proto :HDC, :HBITMAP, :dword, :dword, :ptr, :ptr
                  proto :HDC, :HGDIOBJ
SelectObject
    option dllimport:none
WinMain proto : HINSTANCE, : HINSTANCE, : LPSTR, : UINT
    .data
ClassName db "SimpleWinClass", 0
AppName db "Bitmap rendering", 0
    .data?
hInstance HINSTANCE ?
hBitmap
         HBITMAP ?
CommandLine LPSTR ?
    . code
WinMainCRTStartup proc FRAME
    invoke GetModuleHandleA, NULL
   mov
          hInstance, rax
    invoke GetCommandLineA
          CommandLine, rax
    invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT
    invoke ExitProcess, eax
WinMainCRTStartup endp
WinMain proc FRAME hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPS
    local wc:WNDCLASSEXA
```

```
local msq:MSG
    local hwnd:HWND
    invoke LoadBitmapA, hInst, IDR_BITMAP1
          hBitmap, rax
    mov
    mov
          wc.cbSize, sizeof WNDCLASSEXA
    mov
          wc.style, CS_HREDRAW or CS_VREDRAW
    lea
         rax, [WndProc]
    mov
         wc.lpfnWndProc, rax
          wc.cbClsExtra, NULL
    mov
    mov
         wc.cbWndExtra, NULL
    mov
        rcx, hInst
         wc.hInstance, rcx
    mov
         wc.hbrBackground, COLOR WINDOW+1
    mov
         wc.lpszMenuName, IDR_MENU1
    mov
         rax, [ClassName]
    lea
          wc.lpszClassName, rax
    mov
    invoke LoadIconA, NULL, IDI_APPLICATION
          wc.hIcon, rax
    mov
    mov
          wc.hIconSm, rax
    invoke LoadCursorA, NULL, IDC_ARROW
          wc.hCursor, rax
    mov
    invoke RegisterClassExA, addr wc
    invoke CreateWindowExA, NULL, ADDR ClassName, ADDR AppName,
           WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
           CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
           hInst, NULL
          hwnd, rax
    invoke ShowWindow, hwnd, SW_SHOWNORMAL
    invoke UpdateWindow, hwnd
    .while (1)
        invoke GetMessageA, ADDR msg, NULL, 0, 0
        .break .if (!eax)
        invoke TranslateMessage, ADDR msg
        invoke DispatchMessageA, ADDR msg
    .endw
          rax, msg.wParam
    mov
    ret
WinMain endp
WndProc proc FRAME hWnd: HWND, uMsq:UINT, wParam: WPARAM, lParam: LPARA
local hdc2:HDC
local ps:PAINTSTRUCT
local rect:RECT
local bmi:BITMAPINFOHEADER
```

```
.if edx == WM_DESTROY
        invoke PostQuitMessage, NULL
        xor rax, rax
    .elseif edx == WM_COMMAND
        .if wParam == IDM_EXIT
            invoke DestroyWindow, hWnd
        .endif
        xor eax, eax
    .elseif edx == WM_PAINT
        invoke BeginPaint, hWnd, addr ps
        invoke CreateCompatibleDC, ps.hdc
        mov hdc2, rax
        invoke SelectObject, hdc2, hBitmap
        mov bmi.biSize, sizeof BITMAPINFOHEADER
        mov bmi.biBitCount, 0
        invoke GetDIBits, hdc2, hBitmap, 0, 0, 0, addr bmi, DIB_RGB_
        invoke GetClientRect, hWnd, addr rect
        mov r8d, rect.right
        sub r8d, bmi.biWidth
        jnc @F
        xor r8d, r8d
@@:
        shr r8d, 1
        mov r9d, rect.bottom
        sub r9d, bmi.biHeight
        jnc @F
        xor r9d, r9d
@@:
        shr r9d, 1
        invoke BitBlt, ps.hdc, r8d, r9d, bmi.biWidth, bmi.biHeight,
        invoke DeleteDC, hdc2
        invoke EndPaint, hWnd, addr ps
        xor eax, eax
    .else
        invoke DefWindowProcA, rcx, edx, r8, r9
    .endif
    ret
WndProc endp
if 1 ; for -pe
RT_BITMAP equ 2
RT MENU
          equ 4
;--- menu resource flags
```

```
MF POPUP
         eau 10h
MF_END
         equ 80h
IMAGE_RESOURCE_DIRECTORY struct
Characteristics
                 dword ?
TimeDateStamp
                  dword ?
MaiorVersion
                  word ?
MinorVersion
                  word
NumberOfNamedEntries word ?
NumberOfIdEntries
                  word
IMAGE RESOURCE DIRECTORY ends
IMAGE RESOURCE DIRECTORY ENTRY struct
union
r0
       record NameIsString:1, NameOffset:31
Name
       dword
              ?
       word
Ιd
ends
union
OffsetToData dword
                     DataIsDirectory:1, OffsetToDirectory:31
r1
           record
ends
IMAGE_RESOURCE_DIRECTORY_ENTRY ends
IMAGE_RESOURCE_DATA_ENTRY struct
OffsetToData dword ?
           dword ?
Size
           dword ?
CodePage
Reserved
           dword ?
IMAGE_RESOURCE_DATA_ENTRY ends
   option dotname
.rsrc segment dword FLAT public read 'RSRC'
;--- define menu IDR MENU1 and bitmap IDR BITMAP1
;--- root level: enum the resource types
     IMAGE_RESOURCE_DIRECTORY <0,0,0,0,0,2>
     IMAGE_RESOURCE_DIRECTORY_ENTRY < <RT_BITMAP>, <SECTIONREL bms</pre>
     ;--- second level: enum the IDs of resource type X
bms
     IMAGE RESOURCE DIRECTORY <0,0,0,0,0,1>
     IMAGE RESOURCE DIRECTORY ENTRY < <IDR BITMAP1>, <SECTIONREL bm
menus IMAGE RESOURCE DIRECTORY <0,0,0,0,0,1>
```

```
;--- third level: enum the languages of ID X
      IMAGE RESOURCE DIRECTORY <0,0,0,0,0,1>
      IMAGE_RESOURCE_DIRECTORY_ENTRY < <409h>, <SECTIONREL bm1_l1> >
menu1 IMAGE RESOURCE DIRECTORY <0,0,0,0,0,1>
      IMAGE_RESOURCE_DIRECTORY_ENTRY < <409h>, <SECTIONREL m1_l1> >
;--- last level: define the resource data
;--- data for menu IDR_MENU1, language 409h
m1_l1 IMAGE_RESOURCE_DATA_ENTRY <IMAGEREL m1_l1_data, size_m1_l1, 0,
m1 l1 data dw 0,0 ;menu header
    dw MF_POPUP or MF_END, '&','F','i','l','e',0
    dw MF_END, IDM_EXIT, 'E','&','x','i','t',0
size_m1_l1 equ $ - m1_l1_data
    align 4
;--- data for bitmap IDR_BITMAP1
bm1_l1 IMAGE_RESOURCE_DATA_ENTRY <IMAGEREL bm1_l1_data, size_bm1_l1,</pre>
bm1 l1 data label word
    incbin <Win32_8.bmp>,14     ;skip bitmap file header
size_bm1_l1 equ $ - ( bm1_l1_data )
.rsrc ends
;--- set /subsystem:windows
;--- the PE header is stored in section .hdr$2
    option dotname
.hdr$2 segment dword FLAT public 'HDR'
    org 5Ch ;position to IMAGE_NT_HEADER64.OptionalHeader.Subsystem
            ;2=subsystem windows
.hdr$2 ends
endif
end WinMainCRTStartup
```

Appendix C. Errors and Warnings

The warning and error numbers emitted by JWasm differ from Masm's. However, the texts of the messages are pretty much identical. As in Masm, the first digit shows the severity of the issue: range 1xxx reports a fatal error (assembly process is stopped), range 2xxx indicates an error (assembly process continues, but no object module will be created) and 4xxx are warnings.

x029	Multiple base registers not allowed	In 16-bit code, one cannot use both BX and BP in indirect addressing; in 32/64-bit code, ESP/RSP can only appear once in indirect addressing.
x030	Instruction or register not accepted in current CPU mode	
x031	Invalid addressing mode with current CPU setting	
x032	Cannot use TRn- TRn with current CPU setting	The TRx special registers were restricted to 80386 and 80486 cpus.
x033	Must be index or base register	
x034	Multiple index registers not allowed	
x035		
x036	Scale factor must be 1, 2, 4 or 8	
x037	Cannot be used as index register: <register></register>	Index registers are restricted. In 16-bit mode, only SI and DI can be index registers. In 32-bit mode, all general-purpose registers except ESP can be index registers.
	Base and index	

x038	register differ in	
~ 0 20	Size	
	Expecting comma ORG needs a	
x040	constant or local offset	
x041	POP CS is not allowed	
x042	Only MOV can use special register	The special registers CRx, DRx and TRx can only be moved to/from general purpose registers.
x043	Cannot use SHORT with CALL	Distance of CALL operands must be NEAR or FAR.
x044	Only SHORT jump distance is allowed	Some jump instructions accept short distances only (JCXZ, JECXZ, LOOPx).
x045	Syntax error	
x046	Prefix must be followed by an instruction	
x047	Syntax error: Unexpected colon	
	Operands must be the same size: <size op1=""> - <size op2=""></size></size>	
XU49	Invalid instruction operands	
x050	-	Jcc instructions won't accept type coercions that increase the jump distance (i.e. "jz NEAR PTR <i>label</i> ") if current cpu is < 80386.
XU5T	Immediate data out of range	
	Can not use short or near modifiers	

	with this instruction	
x053	Jump out of range by <num> byte(s)</num>	A short distance must be in the range -128 to +127.
x054	Displacement out of range: <displacement></displacement>	
x055	Initializer value too large	
x056	Symbol already defined: <symbol></symbol>	
	Offset magnitude too large for specified size	
	Magnitude of offset exceeds 16 bit	
x059	Operand 2 too big	
x060	Operand 1 too small	
x061	Line too long	Size of a line (after concatenation) is restricted to 600.
x062	Too many tokens in a line	The number of tokens in a line is restricted to 150.
x063		
x064	Operand is expected	
	-	A constant (numeric) value is expected in the current context. Note that a label - more exactly: the offset part of a label's address - is not a constant value, since the final value is calculated by the linker (or the OS loader) only.
x066	Constant operand is expected	The expression evaluator accepts a constant only in the current context.
	.ELSE clause	An .IF block may contain 0 or 1 .ELSE clauses and it

	already occured in this .IF block	must be the last clause before .ENDIF.		
x068	Multiple overrides			
	Segment, group or segment register expected	The operand before the colon operator (:) must be a segment, group or segment register.		
x070	Identifier too long	Identifer names are restricted to 247. This is a hard limit for OMF output format. For other formats, the limit may be extended by adjusting and recompiling the source code.		
x071	Invalid operand size for instruction			
x072	Not supported: <directive></directive>	Message is displayed if one of the follwing options is specified: OPTION READONLY, OPTION EXPR16, OPTION OLDMACROS. Those are currently not supported.		
x073	Size not specified, assuming: <type></type>	this is a warning. <type> may be BYTE, WORD or DWORD. The message may occur if an immediate value is written to an untyped memory reference: mov [ebx], 1 JWasm makes a guess and displays the warning, while Masm will display an error in such cases.</type>		
x074	Floating-point initializer ignored			
x075	-	Conditional jump (Jcc) instruction destination cannot be far.		
x076	Initializer magnitude too large for specified size			
	Segment attribute			

x077	is defined already: <attribute></attribute>	
	Segment definition changed: %s, %s	
	Class name too long	Segment class names are restricted to 255 in size.
	Block nesting error: %s	
	Segment attribute is unknown: %s	
x082	Must be in segment block	Instructions and directives that generate code or data must be inside a segment block.
	Segment not defined: <segment></segment>	
x084	Colon is expected	
x085	Invalid qualified type: %s	
~ 086	Qualified type is expected	
x087		
x088	Library name is missing	
v089	Cannot access label through segment registers: <label></label>	
x090	Line too long after expansion: <line></line>	
x091	Language type must be specified	
√ 002	PROC, MACRO or macro loop directive must	

	precede LOCAL	
	_	
x093	Cannot nest procedures	
x094	VARARG requires C calling convention	
x095	Multiple .MODEL directives, .MODEL ignored	
x096	Model is not declared	Without a model, simplified segment directives (.CODE, .DATA, .CONST, .DATA?, .STACK, .FARDATA and .FARDATA?) and directives .STARTUP, .EXIT cannot be used.
x097	Backquote missing: ` <identifier></identifier>	
x098	COMMENT delimiter expected	
x099	END directive required at end of file	
XIUU	deep	
x101	Macro nesting level too deep	
x102	Symbol not defined : <symbol></symbol>	
x103		
x104	No filename specified.	
x105	Out of Memory	This is a fatal error. With the 8086-version of jwasm, JWASMR, you'll see this error if you try to assemble something that contains a few thousand symbols. The 32- or 64-bit versions of jwasm should always have

		enough memory on modern machines.
x106	Cannot open file: " <file>" [<error code="">]</error></file>	Error code ENOENT means "file not found". Other error codes are displayed as numbers
x107	Cannot close file: <file> [<error code>]</error </file>	
x108	File write error: <file> [<error code>]</error </file>	Usually happens if output media is read-only or full.
x109	Invalid command- line option: <option></option>	
x110	Internal error in <source file=""/> (<line>)</line>	This error shouldn't be seen in the release version. It's displayed if the internal assert() function is called, which usually is done when a "virtually impossible" error condition has occurred.
	Expecting closing square bracket	
x112	Expecting file name	
	Too many errors	Use <u>commandline option -e</u> to set the max. number of errors that are displayed
x114	forced error <message></message>	Generic "forced error" message
x115	forced error: Value not equal to 0: <value> <text></text></value>	Error emitted by the .ERRNZ directive.
x116	forced error: Value equal to 0: <value> <text></text></value>	Error emitted by the .ERRE directive.
x117	forced error: symbol defined: <symbol></symbol>	Error emitted by the .ERRDEF directive.
	forced error: symbol not	

x118	defined: <symbol></symbol>	Error emitted by the .ERRNDEF directive.
x119	forced error: string blank: <string></string>	Error emitted by the .ERRB directive.
x120	forced error: Ostring not blank: <string></string>	Error emitted by the .ERRNB directive.
x12 2	forced error: strings not equal: <string>: <string></string></string>	Error emitted by the .ERRDIF and .ERRDIFI directives.
	forced error: strings equal: <string>: <string></string></string>	Error emitted by the .ERRIDN and .ERRIDNI directives.
x12 3	<pre>S <file>(<line>): Included by</line></file></pre>	Additional error information if error occured in an include file.
x124	<file>(<line>) ¶[<macro>]: Macro called from</macro></line></file>	Additional error information if error occured inside a macro.
	<file>(<line>): iteration <iteration>: Macro called from</iteration></line></file>	Additional error information if error occured inside a loop macro (FOR, FORC, REPEAT,).
x120	<pre>S <file>(<line>): Main line code</line></file></pre>	Additional error information if error occured inside an include file or a macro.
x12'	Extending jump	
	Directive ignored: %s	
x129	number must be a power of 2	
x130	Incompatible with segment alignment: %s	

x131	Segment expected: %s	
x132	Incompatible CPU mode for 32- bit segment	
	Far call is converted to near call.	
x134	CPU option %s is not valid for selected CPU.	
	Segment '%s' is in another group already	
x136	Symbol type conflict: %s	
x137	Conflicting parameter definition: %s	
x138	PROC and PROTO calling convention conflict	
x139	Non-benign %s redefinition: %s	
x140	Too many bits in RECORD: %s	
X141	Statement not allowed inside structure definition	
	Unmatched block nesting: %s	
x143	Symbol redefinition: %s	
x144	Text item required	

INIVOIZE	
argument type mismatch: argument %u	
Too few arguments to INVOKE: %s	
VARARG parameter must be last	
LABEL parameter must be first	
Too many arguments in macro call: %s	This is a warning. Macro is invoked with more arguments than expected.
Missing operator in expression	
found in	Literals enclosed in <> or {} are items processed by the preprocessor or to initialize "structured" data items. If they're used otherwise, this error will occur.
Initializer must be a string or single item: %s	
Too many initial values for structure: %s	
Too many initial values for array: %s	
String or text literal too long	
PROLOGUE must be macro function	The user-defined prologue macro must be a macro function, that is, there must be an EXITM somewhere inside that returns a literal.
EPILOGUE must be macro procedure: %s	The user-defined epilogue macro must be a macro procedure, that is, there must NOT be an EXITM somewhere inside that returns a literal.
	mismatch: argument %u Too few arguments to INVOKE: %s VARARG parameter must be last LABEL parameter must be first Too many arguments in macro call: %s Missing operator in expression Unexpected literal found in expression: %s Initializer must be a string or single item: %s Too many initial values for structure: %s Too many initial values for array: %s String or text literal too long PROLOGUE must be macro function EPILOGUE must be macro

x158	Reserved word expected		
x159	INVOKE requires prototype for procedure		
x160	Invalid type for data declaration: %s		
	Operand must be RECORD type or field		
	Unmatched macro nesting		
x16 3	Empty (null) string		
x16 4	No segment information to create fixup: %s		
	Register value overwritten by INVOKE		
	Missing quotation mark in string		
x167	Divide by zero in expression		
x168	General Failure		
	Cannot have implicit far jump or call to near label		
x170	Invalid use of register		
x171	Distance invalid for current segment		

	Initializer magnitude too large: %s	
x173	Cannot add two relocatable labels	
	<symbol_name></symbol_name>	Text macros, macros, stack variables, structure fields or segment/groups cannot be public or external.
x175	Positive value expected	
	FAR not allowed in FLAT model COMM variables	
x177	Too many arguments to INVOKE	
	Directive must appear inside a macro	
x179	Invalid type expression	
x180	Cannot declare scoped code label as PUBLIC: <label></label>	
x181	Invalid radix tag	
x182	Instruction operand must have size	The instruction allows operands with more than just one size, and the wanted size cannot be guessed from the current operands.
x183	Use of register assumed to ERROR	
x184	Instructions and initialized data not supported in <seg_type></seg_type>	<pre><seg_type> may be BSS or AT. Such segments don't have data.</seg_type></pre>

	segments	
v105	Literal expected after '='	
x186	No 4k Page- aligned segments in MS386 OMF	A 4K page-aligned segment isn't compatible with MS OMF format, it's a Phar Lab extension. This warning won't appear in the standard version.
x187		
	Operand must be relocatable	
x189	Constant or relocatable label expected	
x190	[ELSE]IF2/.ERR2 not allowed, single-pass assembler	
x191	Expression too complex for UNTILCXZ	
x192	Operands must be in same segment	
x193	Invalid use of external symbol: <symbol_name></symbol_name>	
x194	For -coff leading underscore required for start label: <start_label></start_label>	
x195	Invalid command- line value, default is used: %s	
x196	Unknown fixup type: %u at <segment>. <offset></offset></segment>	

x197	Unsupported fixup type for <format>: <type></type></format>	
x198	Invalid fixup type for <format> <type> at location <segment>. <offset></offset></segment></type></format>	
x199	Syntax error in control-flow directive	
x200	Invalid .model parameter for flat model	
	Output format doesn't support externals: <symbol></symbol>	In formats BIN and MZ all references must be local to the module.
x202	Invalid start label for -bin	
x203	No start label defined	Warning, format MZ only: MZ-binaries usually have a start label. In some cases (i.e. overlays) a missing start label may be ok.
x204	No stack defined	Warning, format MZ only: MZ-binaries usually have a stack. In some cases (i.e. overlays) a missing stack may be ok.
x205	Invalid alignment - value must be 2^n (n=415)	
	Index value past end of string: <value></value>	The index argument of SUBSTR or INSTR is beyond the string argument length
	Count value too large	
x208	Count must be positive or zero	

x209	Syntax error: <item></item>	The parser found an item that has no meaning in the current context
x210		
x211		
	Must use floating- point initializer	
x213	ORG directive not allowed in unions	
	Struct alignment must be 1, 2, 4, 8, 16 or 32	
x215	Structure cannot be instanced	A structure that has size 0 or that contains an ORG directive cannot be instanced.
x216	Missing angle bracket or brace in literal	
x217	Nondigit in number: <number></number>	
	16bit fixup for 32bit label: %s	
x219	Too many macro placeholders	The number of parameters and locals for a macro must not exceed 256
	Missing macro argument: %s, parameter %u	
x221	Doesn't work with 32-bit segments: <directive></directive>	Directives .STARTUP and .EXIT work for 16-bit only.
x222	Segment exceeds 64k limit: %s	In MZ format, 16-bit segments are restricted to 64k.
	Not supported with OMF format: %s	
	Not supported	The directive or feature isn't supported by all formats.

x224		For example, segment-related directives or attributes won't make much sense for flat formats like COFF or ELF.
x225	Unknown default prologue argument: %s	
x226	LOADDS ignored in flat model	
x227	Missing right parenthesis in expression	
x228	Invalid operand for <operator>: <operand></operand></operator>	
x229	Structure improperly initialized: %s	
	Expected: %s	
	Invalid data initializer	
x232	Expected data label	Some operators (LENGTH, SIZE) work with data labels only.
x233	Expression must be a code address	
	-n Option needs a valid name parameter	
x235	Constant value too large: <value></value>	the value of the constant doesn't fit in 64 or - if it is a number to be assigned to a symbolic constant - 32 bits.
x236	Text macro was used before definition	this is a warning only. However, using text macros before they have been defined will force JWasm to do a full second pass, which increases assembly time.
x237	Offset size incompatible with current segment	
	Instruction form	

x238	requires 80386	
x239	Group/Segment offset size conflict: <group offset=""> - <segment offset=""></segment></group>	Segments within a group must all have the same offset size.
x240	Assembly passes reached: <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	Although this is a warning only it usually indicates a severe problem. The assembler is very probably unable to calculate "final" values of all labels and has to be terminated by pressing Ctrl-C.
x241	Filename parameter must be enclosed in <> or quotes	The INCBIN directive requires delimiters for its filename.
x242	Start address on END directive ignored with .STARTUP	
	Invalid symbol type in expression: <symbol></symbol>	The expression evaluator has encountered a symbol that is meaningless in expressions, for example a (text) macro.
x244	Missing right parenthesis	
	Directive must be in control block	.ELSE, .ELSEIF and .ENDIF are valid inside .IF-blocks, .BREAK, .CONTINUE may occur inside .WHILE- or .REPEAT-blocks, .ENDW needs a preceding .WHILE and .UNTIL needs a preceding .REPEAT.
x246	Expected: memory model	the .MODEL directive needs at least one parameter, the memory model.
x247	Type is wrong size for register	
x248	IF[n]DEF expects a plain symbol as argument	this is a warning. Masm accepts any expression as argument for directives [ELSE]IF[N]DEF, but the result probably isn't always what has been expected.

x249	Jump destination must specify a label	
	Ignored: <attribute></attribute>	An attribute or parameter of a directive was found, but not handled. See <u>Known Bugs and missing Features</u> for details about what features aren't implemented yet.
x251	Missing argument for cmdline option	
	Invalid coprocessor register	
x253	Registers AH-DH may not be used with SPL-DIL or R8-R15	64-bit only.
x254	.ENDPROLOG found before EH directives	64-bit only.
x255	Missing FRAME in PROC, no unwind code will be generated	64-bit only.
	Bad alignment for offset in unwind code	64-bit only.
x257	Nonzero value expected	
x258	Size of prolog too big, must be < 256 bytes	64-bit only.
x259	Missing .ENDPROLOG: %s	64-bit only.
x260	.SAFESEH argument must be a PROC	

x261	Directive ignored without -%s switch	
	ELF GNU extensions (8/16-bit relocations) used	This is a warning only. The extensions are not "official", but the GNU linker LD will understand them.
	Syntax error in expression	
x264	Macro label not defined: %s	The target of a GOTO must be within the very same macro.
x265	Procedure argument or local not referenced: %s	This warning is displayed only if at least -W3 is specified.
x266	Group definition too large, truncated: <group_name></group_name>	The size of the OMF record that is to define a group would exceed 4 kB. However, to see this error you'll have to define a group that is to comprise more than 1000 segments,
	COMM variable exceeds 64K: <variable></variable>	in 16-bit, the size of a COMM variable is restricted to 64 kB.
x268	Must be public or external: %s	Names that are to be known by the linker must be public or external.
x269	parameter/local name is reserved word: %s	This is a warning. The names of macro parameters or locals have highest priority and hence the reserved word will become inaccessible within the macro.
x270	real or BCD number not allowed	
XZ/1	expected	
x272	Constant value too large: <value></value>	the value of the constant doesn't fit in 64 or - if it is a number to be assigned to a symbolic constant - 32 bits.
x273	ELSE clause already occured in	An IF block may contain 0 or 1 ELSE clauses and it must be the last clause before ENDIF.

	this IF block	
x274	Illegal use of segment register	
x275	Group exceeds 64K: <group></group>	MZ format only: a group that contains 16-bit segments cannot be larger than 64 kB, because the group must fit into a physical segment.
x276	EXPORT must be FAR: <symbol></symbol>	A 16-bit procedure that is to be exported must be declared with the FAR distance attribute.

Appendix D. Differences between Masm 6 and Masm 8

This is not strictly JWasm-related, but since a lot of documentation that can be found about Masm still refers to Masm v6 or Masm v6.1 only, it's useful to list the differences.

- default output format has been changed from OMF to COFF in Masm 8.
- expression evaluator calculations are done in 64-bit instead of 32-bit. This is not always reflected in the listing file, where the values may be truncated to 32-bit.
- some segment attributes have been added in Masm 8 to cover the COFF section attribute bits. These are:
 - INFO
 - DISCARD
 - NOCACHE
 - NOPAGE
 - SHARED
 - EXECUTE
 - READ
 - WRITE
 - ALIAS
 - ALIGN
- the -safeseh commandline option and the .SAFESEH directive have been added in Masm 8.
- Masm 8 understands a few new data types:
 - OWORD 128-bit integer
 - MMWORD 64-bit data type for MMX
 - XMMWORD 128-bit data type for SSE
- several unary operators are new for Masm 8:
 - LOW32 returns the lower 32-bit of an expression
 - HIGH32 returns the upper 32-bit of an expression
 - IMAGEREL returns the image relative offset of an expression (aka RVA). COFF format only.
 - SECTIONREL returns the section relative offset of an expression.

COFF format only.

Please note: LOW32 and HIGH32 operators won't accept plain numbers with values that exceed 32-bits.

- Masm 8 understands instruction sets up to SSE3. The last version of Masm 6, version 6.15, supports SSE2 only. Masm 8 also understands Intel's VMX and AMD's SVM extensions (x86 virtualization).
- Masm 8 needs at least Windows 2000, it won't run on Windows 9x (without hacks).
- Masm 8 needs a MS VC runtime dll MSVCR80.DLL to run, while Masm 6 was a stand-alone binary.
- Finally, Masm 8 fixes some bugs and introduces a few new ones. One of the new bugs is that Masm 8 has problems with wildcards in filenames; entering

ml.exe -c *.asm

to assemble all assembly files in a directory won't work as expected.

Appendix E. Restrictions of precompiled 16-bit Binary JWASMR.EXE

JWASMR lacks some features of the other precompiled binaries:

- no support for long filenames.
- no support for Open Watcom's fastcall register convention.
- supports OMF, BIN and MZ <u>output formats</u> only.
- no support for 64-bit, SSSE3, SSE4 and AVX.
- no support for OPTION variants that are specific for COFF, ELF or 64-bit.
- no support for **OPTION CODEVIEW** directive.
- no support for directive LABEL's <u>array size syntax extension</u>.

Since the JWasm source is available, one may enable this or that feature if really needed, though.

Appendix F. Additional Features of JWasm's Debug Version

Note: there are no precompiled debug versions of JWasm available. Such versions have to be created from the source code. See the comments in the makefiles how to do that.

The debug version offers the following additional commandline options:

-af	Display all files used in assembly process
-ce	Cause an exception
-dm	Display all messages
-dr	Display reserved words
-drh	Display reserved words hash table
-ds	Display global symbols
-dsh	Display global symbols hash table
-dt	Display debug trace
-ls	Display preprocessed line storage
-nbp	Disable back-patching
-nfp	Do full subsequent passes (disables "fastpass")
-pm= <n></n>	Stop assembly after <i>n</i> passes
-sp	Skip preprocessor step

OPTION CSTACK

```
OPTION CSTACK: [ ON | OFF ]
```

The CSTACK option control the stack-frame creation by <u>invoke</u>. The default setting is OFF.

Example

```
OPTION CSTACK: ON

cstack PROC USES esi edi ebx arg
sub esp,arg
ret
cstack ENDP

OPTION CSTACK: OFF

astack PROC USES esi edi ebx arg
sub esp,arg
ret
astack ENDP
```

Generated code

```
cstack PROC USES esi edi ebx arg
push
        esi
push
        edi
push
        ebx
        ebp
push
        ebp, esp
mov
sub
        esp, arg
leave
        ebx
pop
        edi
pop
        esi
pop
retn
cstack ENDP
```

OPTION CSTACK: OFF

```
astack PROC USES esi edi ebx arg
push
        ebp
mov
        ebp, esp
push
        esi
push
push
sub
        edi
        ebx
        esp, arg
        ebx
pop
        edi
pop
pop
        esi
leave
retn 4
astack ENDP
```

See Also

OPTION SWITCH

OPTION SWITCH: [C | PASCAL | TABLE | NOTABLE | NOREGS | REGAX | NOTE

The <u>switch</u> comes in two main types: a structured switch (Pascal) or an unstructured switch (C). The default type is unstructured.

The TABLE and NOTABLE options control the jump-table creation in the switch. The default setting is NOTABLE.

The NOREGS and REGAX options control the usage of registers in jump-table creation in the switch. The default setting is NOREGS.

The NOTEST option is short lived and skips the range-test in the jump code. The option is turned off after usage.

See Also

OPTION WSTRING

```
OPTION WSTRING:[ ON | OFF ]
```

This toggles ascii/unicode string creation of "quoted strings".

Unicode strings may be used in the <code>@CStr()</code> macro, in function calls, or decleared using:

```
DW "string", 0
```

The default value is OFF. The command-line switch /ws turns this option ON.

See Also

Asmc Fatal Error Messages

A1000 cannot open file : filename
A1001 I/O error closing file : filename
A1002 I/O error writing file : filename
A1005 assembler limit : macro parameter name table full
A1006 invalid command-line option: option
A1007 nesting level too deep
A1008 unmatched macro nesting
A1009 line too long
A1010 unmatched block nesting:
A1011 directive must be in control block
A1012 error count exceeds 100; stopping assembly
A1017 missing source filename
A1901 Internal Assembler Error

See Also

Asmc Error Messages

ASMC Nonfatal Error Messages

ML compatible errors

- ML Nonfatal Error A2004
- ML Nonfatal Error A2005
- ML Nonfatal Error A2006
- ML Nonfatal Error A2007
- ML Nonfatal Error A2008
- **ML Nonfatal Error A2009**
- **ML Nonfatal Error A2010**
- ML Nonfatal Error A2011
- ML Nonfatal Error A2012
- ML Nonfatal Error A2013
- **ML Nonfatal Error A2014**
- ML Nonfatal Error A2015
- **ML Nonfatal Error A2016**
- ML Nonfatal Error A2018
- **ML Nonfatal Error A2019**
- ML Nonfatal Error A2022
- ML Nonfatal Error A2023
- ML Nonfatal Error A2024
- ML Nonfatal Error A2025
- **ML Nonfatal Error A2026**
- ML Nonfatal Error A2028
- **ML Nonfatal Error A2029**
- ML Nonfatal Error A2030
- ML Nonfatal Error A2031
- ML Nonfatal Error A2032
- ML Nonfatal Error A2033
- ML Nonfatal Error A2034
- ML Nonfatal Error A2036
- ML Nonfatal Error A2037 ML Nonfatal Error A2039
- ML Nonfatal Error A2041
- ML Nonfatal Error A2043
- ML Nonfatal Error A2045

- ML Nonfatal Error A2046
- ML Nonfatal Error A2047
- ML Nonfatal Error A2048
- ML Nonfatal Error A2050
- ML Nonfatal Error A2051
- ML Nonfatal Error A2052
- ML Nonfatal Error A2053
- ML Nonfatal Error A2054
- ML Nonfatal Error A2055
- ML Nonfatal Error A2056
- ML Nonfatal Error A2057
- ML Nonfatal Error A2058
- ML Nonfatal Error A2059
- ML Nonfatal Error A2060
- ML Nonfatal Error A2061
- ML Nonfatal Error A2062
- ML Nonfatal Error A2063
- ML Nonfatal Error A2064
- ML Nonfatal Error A2065
- NATION OF THE ADDRESS
- ML Nonfatal Error A2066 ML Nonfatal Error A2068
- ML Nonfatal Error A2070
- ML Nonfatal Error A2071
- TVIE TVOITIGUE ETTOT 71207 1
- ML Nonfatal Error A2072
- ML Nonfatal Error A2074
- ML Nonfatal Error A2075
- ML Nonfatal Error A2076
- ML Nonfatal Error A2077
- ML Nonfatal Error A2079
- ML Nonfatal Error A2080
- ML Nonfatal Error A2081
- ML Nonfatal Error A2082
- ML Nonfatal Error A2083
- ML Nonfatal Error A2084
- ML Nonfatal Error A2085
- ML Nonfatal Error A2086
- ML Nonfatal Error A2087
- ML Nonfatal Error A2088
- ML Nonfatal Error A2089

- ML Nonfatal Error A2090
- ML Nonfatal Error A2091
- ML Nonfatal Error A2092
- ML Nonfatal Error A2093
- ML Nonfatal Error A2094
- ML Nonfatal Error A2095
- ML Nonfatal Error A2096
- ML Nonfatal Error A2097
- ML Nonfatal Error A2098
- ML Nonfatal Error A2100
- ML Nonfatal Error A2101
- ML Nonfatal Error A2103
- ML Nonfatal Error A2104
- **ML Nonfatal Error A2105**
- ML Nonfatal Error A2107
- ML Nonfatal Error A2108
- ML Nonfatal Error A2110
- ML Nonfatal Error A2111
- ML Nonfatal Error A2112
- ML Nonfatal Error A2113
- ML Nonfatal Error A2114
- ML Nonfatal Error A2119
- ML Nonfatal Error A2120
- ML Nonfatal Error A2121
- ML Nonfatal Error A2123
- ML Nonfatal Error A2125
- ML Nonfatal Error A2129
- **ML Nonfatal Error A2131**
- ML Nonfatal Error A2132
- ML Nonfatal Error A2133
- ML Nonfatal Error A2136
- ML Nonfatal Error A2141
- ML Nonfatal Error A2142
- ML Nonfatal Error A2143
- TVIE TVOINGUL ENTOT TIET TE
- ML Nonfatal Error A2144
- ML Nonfatal Error A2145
- ML Nonfatal Error A2147
- ML Nonfatal Error A2148
- ML Nonfatal Error A2151

- ML Nonfatal Error A2154
- ML Nonfatal Error A2156
- **ML Nonfatal Error A2157**
- ML Nonfatal Error A2159
- ML Nonfatal Error A2166
- ML Nonfatal Error A2167
- ML Nonfatal Error A2169
- ML Nonfatal Error A2170
- ML Nonfatal Error A2172
- ML Nonfatal Error A2175
- ML Nonfatal Error A2178
- ML Nonfatal Error A2179
- ML Nonfatal Error A2181
- ML Nonfatal Error A2187
- **ML Nonfatal Error A2189**
- ML Nonfatal Error A2190
- ML Nonfatal Error A2199
- ML Nonfatal Error A2200
- ML Nonfatal Error A2202
- ML Nonfatal Error A2206
- ML Nonfatal Error A2214
- **ML Nonfatal Error A2217**

Non ML compatible errors

- ASMC Nonfatal Error A3000
- **ASMC Nonfatal Error A3001**
- ASMC Nonfatal Error A3002
- ASMC Nonfatal Error A3003
- ASMC Nonfatal Error A3004
- **ASMC Nonfatal Error A3005**
- ASMC Nonfatal Error A3006
- ASMC Nonfatal Error A3007
- ASMC Nonfatal Error A3008
- **ASMC Nonfatal Error A3009**
- **ASMC Nonfatal Error A3010**
- **ASMC Nonfatal Error A3011**
- ASMC Nonfatal Error A3012
- **ASMC Nonfatal Error A3013**

ASMC Nonfatal Error A3014
ASMC Nonfatal Error A3015
ASMC Nonfatal Error A3016
ASMC Nonfatal Error A3017
ASMC Nonfatal Error A3018
ASMC Nonfatal Error A3019
ASMC Nonfatal Error A3020
ASMC Nonfatal Error A3021
ASMC Nonfatal Error A3021

See Also

Asmc Error Messages

ASMC Warning Messages

ML compatible warnings

ML Warning A4003

ML Warning A4005

ML Warning A4006

ML Warning A4007

ML Warning A4008

ML Warning A4011

ML Warning A4012

ML Warning A4910

ML compatible level 3 warnings

ML Warning A6003

ML Warning A6004

ML Warning A6005

Non ML compatible warnings

ASMC Warning A8000

ASMC Warning A8001

ASMC Warning A8002

ASMC Warning A8003

ASMC Warning A8004

ASMC Warning A8005

ASMC Warning A8006

ASMC Warning A8007

ASMC Warning A8008

ASMC Warning A8009

ASMC Warning A8010

ASMC Warning A8011

ASMC Warning A8012

ASMC Warning A8013

ASMC Warning A8014

ASMC Warning A8015 ASMC Warning A8017 ASMC Warning A8018 ASMC Warning A8019 ASMC Warning A8020

Non ML compatible level 3 warnings

ASMC Warning A7000
ASMC Warning A7001
ASMC Warning A7002
ASMC Warning A7003
ASMC Warning A7004
ASMC Warning A7005
ASMC Warning A7006
ASMC Warning A7007
ASMC Warning A7008

See Also

Asmc Error Messages

Parsing of labels

All expansions are pre-processed by the assembler and this may expand macros and other directives before labels. If a macro is added at the same line as a label this may fail.

Example

```
foo macro reg
bswap reg
exitm <reg>
endm

do: mov eax,foo(eax)
...
mov ecx,"3210"
jmp do
```

As a result the code produced by the macro will be expanded above the label and thus the jump will fail.

```
bswap ecx
do: mov eax,ecx
```

Asmc will expand the line left to right in this case.

See Also

Asmc Extensions

Expansion of macros

The <u>label issue</u> becomes a problem in <u>the HLL section</u> where labels are created later:

```
.WHILE macro(...)
```

Asmc will for this reason delay expansion of macros in some of the HLL directives until labels are created. This include .WHILE, .ELSEIF, and .CASE.

See Also

Asmc Extensions

The invoke directive

In Asmc a macro is handled at the same level as a procedure. The header file may then control the expansion:

This is achieved by simply excluding *invoke* as appose to allow invocations of macros.

```
strlen( esi )
```

Asmc sees the combination of a procedure followed by an open bracket as invoke. Empty brackets will be given special handling if the token in front is not a macro.

```
plabel proto
extern elabel:dword
        .data
dlabel label dword
        .code
clabel:
        call
                ax
        call
                eax
        call
                plabel
        call
                elabel
        call
                dlabel
        call
                clabel
                xlabel
        call
        ax()
        eax()
        plabel()
        elabel()
        dlabel()
        clabel()
        xlabel()
```

xlabel:

This simple solution avoids breaking any existing code with a few exceptions: Masm allows brackets to access memory.

```
.if edx < foo( 1 )
; MASM: cmp edx,foo+1
; ASMC: invoke foo, 1 : cmp edx,eax</pre>
```

So square brackets should be used for accessing memory and round brackets to execute. However, an error must then be issued if Asmc extensions are turned off and labels are accessed using round brackets to ensure compatibility.

The inside of brackets may be recursive used at any length including $\underline{\text{C-strings}}$. However, the return code for a procedure is [R|E]AX so there is a limit with regards to OR/AND testing of nested functions.

```
.if foo( bar( 1 ), 2 ) == TRUE
```

See Also

Asmc Extensions

The HLL section

Asmc extensions are allowed in the following conditional control flow directives:

- <u>.IF</u> expression
- .ELSEIF expression
- <u>.UNTIL</u> expression
- .WHILE expression
- .BREAK .IF expression
- .CONTINUE .IF expression
- **.SWITCH** proc(...)
- .CASE expression
- <u>.ENDC</u> .IF expression
- **.ASSERT** expression

See Also

Asmc Extensions

Handling of strings

Given "quoted strings" may be used as arguments, or in general as a const value, C-strings are limited to be used inside brackets of a procedure.

```
.if fopen( "readme.txt", "rt" )
```

See Also

Asmc Extensions | Symbols Reference

Enhanced vector extension

From version 2.26 Asmc support the Intel AVX-512 instruction set.

This includes 12 new 128-bit registers (XMM16 to XMM31), 12 new 256-bit registers (YMM16 to YMM31), 32 new 512-bit registers (ZMM0 to ZMM31), and 8 new opmask registers (K0 to K7).

The EVEX encoding prefix will be omitted by using an EVEX exclusive instruction or any of the extended SIMD registers. A preceding prefix (**{evex}**) may be used for EVEX encoding of other instructions.

See Also

Asmc Extensions | Instruction Sets

OPTION ASMC

```
OPTION ASMC:[ ON | OFF | 0..255 ]
```

This controls the main <u>extension</u> of Asmc. The default value is ON. The command-line switch <u>/Xc</u> turns this option OFF.

See Also

OPTION LOOPALIGN

```
OPTION LOOPALIGN: [0|2|4|8|16]
```

This controls the alignment for .WHILE and .REPEAT labels. The default value is 0.

```
jmp loop_start

* ALIGN <loopalign> ; align label after jump
loop_label:
```

See Also

OPTION CASEALIGN

OPTION CASEALIGN: [0|2|4|8|16]

This controls the alignment for .CASE labels. The default value is 0.

* ALIGN <casealign> ; align label after jump
case:

See Also

Directives Reference

.IF

```
.IF condition1
    statements
    [[.ELSEIF condition2
        statements]]
    [[.ELSE
        statements]]
    .ENDIF
```

Generates code that tests *condition1* (for example, AX > 7) and executes the *statements* if that condition is true. If a .ELSE follows, its statements are executed if the original condition was false. Note that the conditions are evaluated at run time.

See Also

<u>Directives Reference</u> | <u>Flag conditions</u> | <u>Signed compare</u> | <u>Return code</u>

.WHILE

.WHILE condition statements .ENDW

Generates code that executes the block of *statements* while *condition* remains true.

See Also

<u>Directives Reference</u> | <u>Flag conditions</u> | <u>Signed compare</u> | <u>Return code</u>

.REPEAT

.REPEAT
statements
.UNTIL condition

Generates code that repeats execution of the block of *statements* until *condition* becomes true. <u>.UNTILCXZ</u>, which becomes true when CX is zero, may be substituted for .UNTIL. The *condition* is optional with **.UNTILCXZ**.

See Also

<u>Directives Reference</u> | <u>Signed compare</u> | <u>Return code</u>

.UNTILCXZ

See <u>.REPEAT</u>

See Also

Directives Reference, .UNTILAXZ, .UNTILBXZ, .UNTILDXZ

.BREAK

```
.BREAK(n) [[.IF condition]]
```

Generates code to terminate a .WHILE or .REPEAT block if *condition* is true.

BREAK[(n)] is optional nesting level to terminate.

```
.while 1
                   ; break .while 1
   .break
   .while 2
   .break(1)
               ; break .while 1
   .while 3
       .break(2) ; break .while 1
       .while 4
           .break(3) ; break .while 1
           .break(2) ; break .while 2
           .break(1) ; break .while 3
       .endw
   .endw
   .endw
.endw
```

See Also

Directives Reference | .CONTINUE

.CONTINUE

```
.CONTINUE[([0]1..n)] [[.IF condition]]
```

Generates code to jump to the top of a <u>.WHILE</u> or <u>.REPEAT</u> block if *condition* is true.

.CONTINUE[(1..n)] is optional nesting level to continue.

.CONTINUE[(0[1..n])] jump's directly to START label: no TEST.

See Also

<u>Directives Reference</u> | <u>.BREAK</u>

.FOR

```
.FOR[S] [initialization] : [condition] : [increment/decrement]
    statements
    .ENDF
```

Generates code that executes the block of *statements* while *condition* remains true.

See Also

<u>Directives Reference</u> | <u>Flag conditions</u> | <u>Signed compare</u> | <u>.ENDF</u>

.SWITCH

The switch comes in three main variants: a structured switch, as in Pascal, which takes exactly one branch, an unstructured switch, as in C, which functions as a type of goto, and a control table switch with the added possibility of testing for combinations of input values, using boolean style AND/OR conditions, and potentially calling subroutines instead of just a single set of values.

The control table switch is declared with no arguments and each .CASE directive does all the testing.

The unstructured switch works as a regular C switch where each .CASE directive is just a label.

```
.switch eax
  .case 0: .repeat : movsb
  .case 7: movsb
  .case 6: movsb
  .case 5: movsb
  .case 4: movsb
  .case 3: movsb
  .case 2: movsb
  .case 1: movsb : .untilcxz
.endsw
```

The structured switch works as a regular Pascal switch where each .CASE directive is a closed branch.

```
.switch eax
  .case 1: printf("Gold medal")
  .case 2: printf("Silver medal")
  .case 3: printf("Bronze medal")
  .default
        printf("Better luck next time")
.endsw
```

See Also

Directives Reference

.CASE

Case opens a case statement. The case statement compares the value of an ordinal expression to each selector, which can be a constant, a subrange, or a list of them separated by commas.

The selector field is separated from action field by Colon or a new line.

```
.CASE 1: mov ax,2 : .ENDC
.CASE 2
    mov ax,3
    .ENDC
.CASE al
.CASE 0,1,4,7
.CASE 0..9
```

In the control table switch .CASE is equal to .IF:

```
.CASE al .CASE ax <= 2 && !bx
```

See Also

<u>.SWITCH | .ENDC | .DEFAULT | .ENDSW</u>

.ENDC

.ENDC closes a .CASE statement.

The name was separated from BREAK to have more flexibility with regards to control flow of loops. However, ENDC have the same qualities as BREAK and thus can be used in combination with .IF:

.ENDC .IF al
$$== 2$$

See Also

.SWITCH | .CASE | .DEFAULT | .GOTOSW | .ENDSW

.GOTOSW

```
.GOTOSW[1|2|3] [[(<case_val>)] | [.IF condition]]
```

Generates code to jump to the top of a <u>.SWITCH</u> block if *condition* is true.

.GOTOSW jump's directly to the TEST label.

```
.switch al
  .case 1
  ...
  .case 9
   mov al,1
   .gotosw ; "Jump" to case 1
```

GOTOSW[1|2|3] is optional nesting level to continue.

```
.switch al
  .case 1
                         ; continue .switch al
    .gotosw
    .switch bl
      .case 1
                        ; continue .switch al
        .gotosw1
        .switch cl
          .case 1
                        ; continue .switch al
            .gotosw2
            .switch dl
              .case 1
                .gotosw3 ; continue .switch al
                .gotosw2 ; continue .switch bl
                .gotosw1 ; continue .switch cl
                  Direct jump to .switch cl / case 1
                .gotosw1(1)
            .endsw
        .endsw
    .endsw
.endsw
```

GOTOSW can be used in combination with .**IF** *condition*, or a direct jump to .**GOTOSW**(<*case_val*>).

See Also

<u>Directives Reference</u> | <u>.ENDC</u> | <u>.SWITCH</u>

.DEFAULT

.DEFAULT executes when none of the other cases match the control expression.

See Also

<u>.SWITCH | .CASE | .ENDC | .ENDSW</u>

.ENDSW

.ENDSW closes a .SWITCH statement.

See Also

<u>.SWITCH | .CASE | .ENDC | .DEFAULT</u>

.COMDEF

.COMDEF name

Declares a structure type for a COM interface based on the Component Object Model introduced by Microsoft in 1993.

A typical C declaration for a COM interface:

```
typedef struct IShellFolderVtbl {
    BEGIN_INTERFACE
    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IShellFolder * _This, REFIID riid, void **ppv0bject);
        ...
    END_INTERFACE
    } IShellFolderVtbl;
interface IShellFolder {
    CONST_VTBL struct IShellFolderVtbl *lpVtbl;
    };

Same using .COMDEF:
    .comdef IShellFolder
    QueryInterface proc :REFIID, :ptr
    ...
    .ends
```

The objects are normally instantiated with a CreateInstance() function that takes the class as argument and we end up with a pointer or just AX. The call to the method is then something like this:

```
mov rcx,rax ; load args
mov rdx,riid
mov r8,ppv0bject
mov rax,[rcx] ; make the call
call [rax].IShellFolderVtbl.QueryInterface
```

The assembler needs to know the name of the base class in order to use these methods. If the method name is not found in the base class, 'Vtbl' is added to the class name. If method exist in classVtbl the pointer is assumed to be the first member of the base (lpVtbl).

```
local s:IShellFolder
s.QueryInterface(riid, ppv0bject)
* lea rcx,s

local p:ptr IShellFolder
p.QueryInterface(riid, ppv0bject)
* mov rcx,p

assume rbx:ptr IShellFolder
[rbx].QueryInterface(riid, ppv0bject)
* mov rcx,rbx
```

If PROC is used inside a structure (normally an error) Asmc assumes this to be a pointer to a function.

```
* T$0001 typedef proto WINAPI :REFIID, :ptr

* P$0001 typedef ptr T$0001

* QueryInterface P$0001 ?
```

The first method close the base class and open (in this case) the IShellFolderVtbl structure and .ENDS will then close the current struct.

```
.ends
* IShellFolderVtbl ends
```

See Also

<u>.ENDS | .CLASSDEF</u>

.CLASSDEF

.CLASSDEF name [args]

Declares a structure type for a **COM** interface.

.CLASSDEF adds the following types:

```
.classdef Class
  * LPCLASS typedef ptr Class
  * LPCLASSVtbl typedef ptr ClassVtbl
  * Class@Class proto :ptr Class
  * Class struct 8
  * lpVtbl LPCLASSVtbl ?
```

Release() is added as the first method:

```
Method1 proc local ; static
  Method2 proc :ptr ; first virtual function
* Class ends
* ClassVtbl struct
* T$000B typedef proto :ptr Class
* P$000B typedef ptr T$000B
* Release P$000B ?
* T$000B typedef proto :ptr Class
* P$000B typedef proto :ptr Class
* P$000B typedef ptr T$000B
* Method2 P$000B ?
```

To define a method locally in the base class the keyword LOCAL may be used. Locally defined functions are called directly without the _this argument omitted.

```
assume rcx:LPCLASS

foo proc

local p:LPCLASS

[rcx].Method1()
[rcx].Method2(rdx)
[rcx].Release()
ret
```

foo endp

Code produced:

0:	55				push	rbp	
1:	48	8b	ec		mov	rbp,rsp	
4:	48	83	ec	30	sub	rsp,0x30	
8:	ff	51	10		call	QWORD PTR	[rcx+0x10]
b:	48	8b	01		mov	rax,QWORD	PTR [rcx]
e:	ff	50	80		call	QWORD PTR	[rax+0x8]
11:	48	8b	01		mov	rax,QWORD	PTR [rcx]
14:	ff	10			call	QWORD PTR	[rax]
16:	с9				leave		
17:	с3				ret		

See Also

.ENDS | .COMDEF

.ENDS

.ENDS closes a .COMDEF or .CLASSDEF statement.

See Also

.COMDEF | .CLASSDEF

.ASSERT

.ASSERT[D|W|B] <assert_expression> .ASSERT: [<handler> | ON | OFF | PUSH | POP | PUSHF | POPF | CODE | E

Options

ON/OFF Main switch.

Save and restore the ASMC flag. Stack **PUSH/POP**

level is 128.

PUSHF/POPF Toggles using PUSHF[D|Q] before calling

handler.

CODE/ENDS Assemble code section if ASSERT is ON.

The assert macro calls this routine if

Handler *expression* is not true. The default handler

name is assert exit.

See Also

Directives Reference | .IF | Return code

Flag conditions

The options listed may be used as directive **.IF***xx*, **.WHILE***xx*, and **.UNTIL***xx*.

```
XX
             Above | Not Below or Equal
Α
     NBE
     C/NAE Below | Carry | Not Above or Equal
В
G
            Greater | Not Less or Equal (signed)
     NLE
            Less | Not Greater or Equal (signed)
L
     NGE
             Overflow (signed)
0
P
             Parity | Parity Even
     PE
S
             Signed (signed)
             Zero | Equal
Z
     \boldsymbol{E}
     BE
             Not Above | Not Below or Equal
NA
NB
     NC/AE Not Below | Not Carry | Above or Equal
             Not Greater | Less or Equal (signed)
NG
     LE
             Not Less | Greater or Equal (signed)
NL
     GE
             Not Overflow (signed)
NO
NP
     PO
             No Parity | Parity Odd
NS
             Not Signed (signed)
NZ
     NE
             Not Zero | Not Equal
```

Note that if used with *condition* the directive may have a different meaning.

See Also

<u>Directives Reference</u> | <u>Signed compare</u> | <u>Return code</u>

Signed compare

Flips the expression to signed if the first operand is a register.

.IFSx, .WHILESx, and .UNTILSx <expression>.

XX		
S	Signed REG	
SB	Signed AL	
SW	Signed AX	
SD	Signed EAX	

See Also

Directives Reference | Flag conditions | Return code

Return code

Sets size from a function call to *xx*.

.IFxx, .WHILExx, .UNTILxx, and .ASSERTx <expression>.

ļ	
xx	
В	ВҮТЕ
W	WORD
D	DWORD
SB	SBYTE
SW	SWORD
SD	SDWORD

See Also

<u>Directives Reference</u> | <u>Flag conditions</u> | <u>Signed compare</u>

@Date

@Date

The system date in the format yyyy-mm-dd (text macro).

See Also

Symbols Reference

@CStr

@CStr(string)

Macro function that creates a string in the .DATA segment. The macro accepts C-escape characters in the string. Strings are added to a stack and reused if duplicated strings are found. The macro returns *offset string*.

Example

```
mov eax,@CStr( "\tCreate a \"C\" string: %s%d\n" )
mov ebx,@CStr( "string: %s%d\n" )
mov ecx,@CStr( "%s%d\n" )
mov edx,@CStr( "%d\n" )
mov edi,@CStr( "\n" )
```

Generated code

```
.data
DS0000 db 9,"Create a ",'"',"C",'"'," string: %s%d",10,0
.code
mov eax,offset DS0000
mov ebx,offset DS0000[14]
mov ecx,offset DS0000[22]
mov edx,offset DS0000[24]
mov edi,offset DS0000[26]
```

See Also

Symbols Reference

6.2.1 Win64 Structured Exception Handling (SEH)

SEH in Win64 differs significantly from the implementation in Win32. It's very well possible to ignore Win64 SEH for assembly. However, if an assembly routine wants to comply to these rules, a thorough understanding of the Win64 ABI is necessary. Masm (the 64-bit version) supplies some "primitives" for SEH support (.ALLOCSTACK, .PUSHREG, .SAVEREG, ...), along with a new FRAME attribute for the PROC directive. These features are also supported by JWasm. See sample Win64_3 how the "primitives" are to be used for SEH support.

The big disadvantage is that using the FRAME keyword in Masm "disables" most of the other high level features combined with PROC (function parameters, locals and registers saved with USES) because no function prologues and epilogues are generated anymore. Additionally, the implementation in some Masm versions seems to be a bit buggy. Because of this and to ease the usage of SEH in Win64 there is a new directive implemented in JWasm:

OPTION FRAME: AUTO

If this option is set, JWasm will create Win64 SEH-compatible prologues and epilogues. If the option is off, JWasm will behave Masm-compatible, that is, FRAME found in a PROC directive will disable automatic prologue/epilogue generation. See sample Win64 3e how this option is supposed to be used.

As for the PROC syntax: The Masm documentation states that FRAME can be used in combination with USES and procedure parameters and must be located behind all parameters. However, this syntax isn't accepted by any Masm version. The only syntax which Masm will accept without being confused is FRAME as the one and only parameter for PROC. Therefore JWasm doesn't follow the Masm documentation in this point: the optional FRAME keyword is expected *before* the procedure parameters. The syntax in JWasm is:

```
procname PROC [public] FRAME[:exc_handler] [USES <reglist>] [par
```

The SEH "primitives" will generate some additional data in segments .pdata and .xdata. This data is somewhat hidden, but JWasm will display the corresponding data definitions in the listing if option -Sg is set.

cannot open file : filename

- The file does not exist.
- The file is in use by another process.
- The filename is not valid.
- A read-only file with the output filename already exists.
- The current drive is full.
- The current directory is the root and is full.
- The device cannot be written to.
- The drive is not ready.

See Also

^{*} ML 6.0 compatible error

I/O error closing file

- The operating system returned an error when the assembler attempted to close a file.
- This error can be caused by having a corrupt file system or by removing a disk before the file could be closed.

See Also

^{*} ML 6.0 compatible error

I/O error writing file

The assembler was unable to write to an output file. One of the following may be a cause:

- The current drive is full.
- The current directory is the root and is full.
- The device cannot be written to.
- The drive is not ready.

See Also

^{*} ML 6.0 compatible error

Assembler limit: macro parameter name table full

Too many parameters, locals, or macro labels were defined for a macro. There was no more room in the macro name table.

Define shorter or fewer names, or remove unnecessary macros.

* ML 6.0 compatible error

See Also

Invalid command-line option: option

ASMC did not recognize the given parameter as an option. This error is generally caused when there is a syntax error on the command line.

* ML 6.0 compatible error

See Also

Nesting level too deep

The assembler reached its nesting limit. The limit is 20 levels except where noted otherwise.

One of the following was nested too deeply:

- A high-level directive such as .IF, .REPEAT, or .WHILE
- A structure definition
- A conditional-assembly directive
- A procedure definition
- A PUSHCONTEXT directive (the limit is 10).
- A segment definition
- An include file
- A macro

See Also

^{*} ML 6.0 compatible error

Unmatched macro nesting

Either a macro was not terminated before the end of the file, or the terminating directive ENDM was found outside of a macro block. One cause of this error is omission of the dot before .REPEAT or .WHILE.

* ML 6.0 compatible error

See Also

Line too long

A line in a source file exceeded the limit of 1024 characters.

If multiple physical lines are concatenated with the line-continuation character (\setminus), the resulting logical line is still limited to 1024 characters.

* ML 6.0 compatible error

See Also

Unmatched block nesting:

A block beginning did not have a matching end, or a block end did not have a matching beginning. One of the following may be involved:

- A high-level directive such as .IF, .REPEAT, or .WHILE
- A conditional-assembly directive such as IF, REPEAT, or WHILE
- A structure or union definition
- A procedure definition
- A segment definition
- A POPCONTEXT directive
- A conditional-assembly directive, such as an ELSE, ELSEIF, or ENDIF without a matching IF

See Also

^{*} ML 6.0 compatible error

Directive must be in control block

The assembler found a high-level directive where one was not expected. One of the following directives was found:

- .ELSE without .IF
- .ENDIF without .IF
- .ENDW without .WHILE
- .UNTIL[[CXZ]] without .REPEAT
- .CONTINUE without .WHILE or .REPEAT
- .BREAK without .WHILE or .REPEAT
- .ELSE following .ELSE

See Also

^{*} ML 6.0 compatible error

Error count exceeds 100; stopping assembly

The number of nonfatal errors exceeded the assembler limit of 100. Nonfatal errors are in the range A2xxx. When warnings are treated as errors they are included in the count. Warnings are considered errors if you use the /Wx command-line option, or if you set the Warnings Treated as Errors option in the Macro Assembler Global Options dialog box of PWB.

* ML 6.0 compatible error

See Also

ASMC Fatal Error A1017

Missing source filename

ML could not find a file to assemble or pass to the linker.

This error is generated when you give ASMC command-line options without specifying a filename to act upon. To assemble files that do not have a .ASM extension, use the /Ta command-line option.

This error can also be generated by invoking ASMC with no parameters if the ASMC environment variable contains command-line options.

* ML 6.0 compatible error

See Also

ASMC Fatal Error A1901

Internal Assembler Error

Contact Product Support Services

The ASMC driver, called ASMC.exe, generated a system error. Note the circumstances of the error and file a bug report at <u>ASMC Development</u>

* ML 6.0 compatible error

See Also

Symbol type conflict: identifier

The EXTERNDEF or LABEL directive was used on a variable, symbol, data structure, or label that was defined in the same module but with a different type.

* ML 6.0 compatible error

See Also

Symbol redefinition: identifier

The given nonredefinable symbol was defined in two places.

* ML 6.0 compatible error

See Also

Undefined symbol: identifier

An attempt was made to use a symbol that was not defined. One of the following may have occurred:

- A symbol was not defined.
- A field was not a member of the specified structure.
- A symbol was defined in an include file that was not included.
- An external symbol was used without an EXTERN or EXTERNDEF directive.
- A symbol name was misspelled.
- A local code label was referenced outside of its scope.

See Also

^{*} ML 6.0 compatible error

Non-benign record redefinition

A RECORD definition conflicted with a previous definition. One of the following occurred:

- There were different numbers of fields.
- There were different numbers of bits in a field.
- There was a different label.
- There were different initializers.

See Also

^{*} ML 6.0 compatible error

syntax error:

A token at the current location caused a syntax error. One of the following may have occurred:

- A dot prefix was added to or omitted from a directive.
- A reserved word (such as C or SIZE) was used as an identifier.
- An instruction was used that was not available with the current processor or coprocessor selection.
- A comparison run-time operator (such as ==) was used in a conditional assembly statement instead of a relational operator (such as EQ).
- An instruction or directive was given too few operands.
- An obsolete directive was used.

See Also

^{*} ML 6.0 compatible error

syntax error in expression

An expression on the current line contained a syntax error. This error message may also be a side-effect of a preceding program error.

* ML 6.0 compatible error

See Also

invalid type expression

The operand to THIS or PTR was not a valid type expression.

* ML 6.0 compatible error

See Also

distance invalid for word size of current segment

A procedure definition or a code label defined with LABEL specified an address size that was incompatible with the current segment size. One of the following occurred:

- A NEAR16 or FAR16 procedure was defined in a 32-bit segment.
- A NEAR32 or FAR32 procedure was defined in a 16-bit segment.
- A code label defined with LABEL specified FAR16 or NEAR16 in a 32-bit segment.
- A code label defined with LABEL specified FAR32 or NEAR32 in a 16-bit segment.

* ML 6.0 compatible error

See Also

PROC, MACRO, or macro repeat directive must precede LOCAL

A LOCAL directive must be immediately preceded by a MACRO, PROC, macro repeat directive (such as REPEAT, WHILE, or FOR), or another LOCAL directive.

* ML 6.0 compatible error

See Also

.MODEL must precede this directive

A simplified segment directive or a .STARTUP or .EXIT directive was not preceded by a .MODEL directive. A .MODEL directive must specify the model defaults before a simplified segment directive, or a .STARTUP or .EXIT directive may be used.

* ML 6.0 compatible error

See Also

cannot define as public or external: identifier

Only labels, procedures, and numeric equates can be made public or external using PUBLIC, EXTERN, or EXTERNDEF. Local code labels cannot be made public.

* ML 6.0 compatible error

See Also

segment attributes cannot change: attribute

A segment was reopened with different attributes than it was opened with originally.

When a SEGMENT directive opens a previously defined segment, the newly opened segment inherits the attributes the segment was defined with.

* ML 6.0 compatible error

See Also

expression expected

The assembler expected an expression at the current location but found one of the following:

- A unary operator without an operand
- A binary operator without two operands
- An empty pair of parentheses, (), or brackets, []

See Also

^{*} ML 6.0 compatible error

invalid use of external symbol: identifier

An attempt was made to compare the given external symbol using a relational operator.

The comparison cannot be made because the value or address of an external symbol is not known at assembly time.

* ML 6.0 compatible error

See Also

operand must be RECORD type or field

The operand following the WIDTH or MASK operator was not valid. The WIDTH operator takes an operand that is the name of a field or a record. The MASK operator takes an operand that is the name of a field or a record type.

* ML 6.0 compatible error

See Also

instruction operands must be the same size

The operands to an instruction did not have the same size.

* ML 6.0 compatible error

See Also

instruction operand must have size

At least one of the operands to an instruction must have a known size.

* ML 6.0 compatible error

See Also

invalid operand size for instruction

The size of an operand was not valid.

* ML 6.0 compatible error

See Also

operands must be in same segment

Relocatable operands used with a relational or minus operator were not located in the same segment.

* ML 6.0 compatible error

See Also

constant expected

The assembler expected a constant expression at the current location. A constant expression is a numeric expression that can be resolved at assembly time.

* ML 6.0 compatible error

See Also

expression must be a code address

An expression evaluating to a code address was expected. One of the following occurred:

- SHORT was not followed by a code address.
- NEAR PTR or FAR PTR was applied to something that was not a code address.

* ML 6.0 compatible error

See Also

multiple base registers not allowed

An attempt was made to combine two base registers in a memory expression.

For example, the following expressions cause this error:

```
[bx+bp]
[bx][bp]
```

In another example, given the following definition:

```
id1 proc arg1:byte
```

either of the following lines causes this error:

```
mov al, [bx].arg1
lea ax, arg1[bx]
```

* ML 6.0 compatible error

See Also

multiple index registers not allowed

An attempt was made to combine two index registers in a memory expression.

For example, the following expressions cause this error:

* ML 6.0 compatible error

See Also

must be index or base register

An attempt was made to use a register that was not a base or index register in a memory expression.

For example, the following expressions cause this error:

[ax] [bl]

* ML 6.0 compatible error

See Also

invalid use of register

An attempt was made to use a register that was not valid for the intended use. One of the following occurred:

- OFFSET was applied to a register. (OFFSET can be applied to a register under the M510 option.)
- A special 386 register was used in an invalid context.
- A register was cast with PTR to a type of invalid size.
- A register was specified as the right operand of a segment override operator (:).
- A register was specified as the right operand of a binary minus operator ().
- An attempt was made to multiply registers using the * operator.
- Brackets ([]) were missing around a register that was added to something.

See Also

^{*} ML 6.0 compatible error

invalid INVOKE argument : argument number

The INVOKE directive was passed a special 386 register, or a register pair containing a byte register or special 386 register. These registers are illegal with INVOKE.

* ML 6.0 compatible error

See Also

must be in segment block

One of the following was found outside of a segment block:

- An instruction
- A label definition
- A THIS operator
- A \$ operator
- A procedure definition
- An ALIGN directive
- An ORG directive

See Also

^{*} ML 6.0 compatible error

too many initial values for structure: structure

The given structure was defined with more initializers than the number of fields in the type declaration of the structure.

* ML 6.0 compatible error

See Also

statement not allowed inside structure definition

A structure definition contained an invalid statement. A structure cannot contain instructions, labels, procedures, control-flow directives, .STARTUP, or .EXIT.

* ML 6.0 compatible error

See Also

line too long

A source-file line exceeded the limit of 1024 characters.

If multiple physical lines are concatenated with the line-continuation character (\setminus), the resulting logical line is still limited to 1024 characters.

* ML 6.0 compatible error

See Also

string or text literal too long

A string or text literal, or a macro function return value, exceeded the limit of 255 characters.

* ML 6.0 compatible error

See Also

identifier too long

An identifier exceeded the limit of 247 characters.

* ML 6.0 compatible error

See Also

missing angle bracket or brace in literal

An unmatched angle bracket (either < or >) or brace (either { or }) was found in a literal constant or an initializer. One of the following occurred:

- A pair of angle brackets or braces was not complete.
- An angle bracket was intended to be literal, but it was not preceded by an exclamation point (!) to indicate a literal character.

See Also

^{*} ML 6.0 compatible error

missing single or double quotation mark in string

An unmatched quotation mark (either ' or ") was found in a string. One of the following may have occurred:

- A pair of quotation marks around a string was not complete.
- A pair of quotation marks around a string was formed of one single and one double quotation mark.
- A single or double quotation mark was intended to be literal, but the surrounding quotation marks were the same kind as the literal one.

* ML 6.0 compatible error

See Also

empty (null) string

A string consisted of a delimiting pair of quotation marks and no characters within.

For a string to be valid, it must contain 1-255 characters.

* ML 6.0 compatible error

See Also

nondigit in number

A number contained a character that was not in the set of characters used by the current radix (base).

This error can occur if a B or D radix specifier is used when the default radix is one that includes that letter as a valid digit.

* ML 6.0 compatible error

See Also

real or BCD number not allowed

A floating-point (real) number or binary coded decimal (BCD) constant was used other than as a data initializer.

One of the following occurred:

- A real number or a BCD was used in an expression.
- A real number was used to initialize a directive other than DWORD, QWORD, or TBYTE.
- A BCD was used to initialize a directive other than TBYTE.

See Also

^{*} ML 6.0 compatible error

text item required

A literal constant or text macro was expected.

One of the following was expected:

- A literal constant, which is text enclosed in <>
- A text macro name
- A macro function call
- A % followed by a constant expression

See Also

^{*} ML 6.0 compatible error

forced error

The conditional-error directive .ERR or .ERR1 was used to generate this error.

* ML 6.0 compatible error

See Also

forced error : value equal to 0

The conditional-error directive .ERRE was used to generate this error.

* ML 6.0 compatible error

See Also

forced error : value not equal to 0

The conditional-error directive .ERRNZ was used to generate this error.

* ML 6.0 compatible error

See Also

forced error: symbol not defined

The conditional-error directive .ERRNDEF was used to generate this error.

* ML 6.0 compatible error

See Also

forced error: symbol defined

The conditional-error directive .ERRDEF was used to generate this error.

* ML 6.0 compatible error

See Also

forced error: string blank

The conditional-error directive .ERRB was used to generate this error.

* ML 6.0 compatible error

See Also

forced error : string not blank

The conditional-error directive .ERRNB was used to generate this error.

* ML 6.0 compatible error

See Also

forced error: strings equal

The conditional-error directive .ERRIDN or .ERRIDNI was used to generate this error.

* ML 6.0 compatible error

See Also

forced error : strings not equal

The conditional-error directive .ERRDIF or .ERRDIFI was used to generate this error.

* ML 6.0 compatible error

See Also

[[[ELSE]]]IF2/.ERR2 not allowed : single-pass assembler

A directive for a two-pass assembler was found.

The Microsoft Macro Assembler (MASM) is a one-pass assembler. MASM does not accept the IF2, ELSEIF2, and .ERR2 directives. This error also occurs if an ELSE directive follows an IF1 directive.

* ML 6.0 compatible error

See Also

expression too complex for .UNTILCXZ $\,$

An expression used in the condition that follows .UNTILCXZ was too complex. The .UNTILCXZ directive can take only one expression, which can contain only == or !=. It cannot take other comparison operators or more complex expressions using operators like \parallel .

* ML 6.0 compatible error

See Also

can ALIGN only to power of 2: expression

The expression specified with the ALIGN directive was invalid. The ALIGN expression must be a power of 2 between 2 and 256, and must be less than or equal to the alignment of the current segment, structure, or union.

* ML 6.0 compatible error

See Also

structure alignment must be 1, 2, or 4

The alignment specified in a structure definition was invalid.

* ML 6.0 compatible error

See Also

expected: token

The assembler expected the given token.

* ML 6.0 compatible error

See Also

incompatible CPU mode and segment size

An attempt was made to open a segment with a USE16, USE32, or FLAT attribute that was not compatible with the specified CPU, or to change to a 16-bit CPU while in a 32bit segment.

The USE32 and FLAT attributes must be preceded by one of the following processor directives: .386, .386C, .386P, .486, or .486P.

* ML 6.0 compatible error

See Also

instruction prefix not allowed

One of the prefixes REP, REPE, REPNE, or LOCK preceded an instruction for which it was not valid.

* ML 6.0 compatible error

See Also

invalid instruction operands

One or more operands were not valid for the instruction they were specified with.

* ML 6.0 compatible error

See Also

initializer too large for specified size

An initializer value was too large for the data area it was initializing.

* ML 6.0 compatible error

See Also

cannot access symbol in given segment or group: identifier

The given identifier cannot be addressed from the segment or group specified.

* ML 6.0 compatible error

See Also

cannot access label through segment registers

An attempt was made to access a label through a segment register that was not assumed to its segment or group.

* ML 6.0 compatible error

See Also

jump destination too far [: by 'n' bytes]

The destination specified with a jump instruction was too far from the instruction. One of the following may be a solution:

- Enable the LJMP option.
- Remove the SHORT operator. If SHORT has forced a jump that is too far, n is the number of bytes out of range.
- Rearrange code so that the jump is no longer out of range.

See Also

^{*} ML 6.0 compatible error

jump destination must specify a label

A direct jumps' destination must be relative to a code label.

* ML 6.0 compatible error

See Also

instruction does not allow NEAR indirect addressing

A conditional jump or loop cannot take a memory operand. It must be given a relative address or label.

* ML 6.0 compatible error

See Also

instruction does not allow FAR direct addressing

A conditional jump or loop cannot be to a different segment or group.

* ML 6.0 compatible error

See Also

jump distance not possible in current CPU mode

A distance was specified with a jump instruction that was incompatible with the current processor mode.

For example, 48-bit jumps require .386 or above.

* ML 6.0 compatible error

See Also

missing operand after unary operator

An operator required an operand, but no operand followed.

* ML 6.0 compatible error

See Also

cannot mix 16- and 32-bit registers

An address expression contained both 16- and 32-bit registers. For example, the following expression causes this error:

[bx+edi]

* ML 6.0 compatible error

See Also

invalid scale value

A register scale was specified that was not 1, 2, 4, or 8.

* ML 6.0 compatible error

See Also

constant value too large

A constant was specified that was too big for the context in which it was used.

* ML 6.0 compatible error

See Also

instruction or register not accepted in current CPU mode

An attempt was made to use an instruction, register, or keyword that was not valid for the current processor mode.

For example, 32-bit registers require .386 or above. Control registers such as CR0 require privileged mode .386P or above. This error will also be generated for the NEAR32, FAR32, and FLAT keywords, which require .386 or above.

* ML 6.0 compatible error

See Also

reserved word expected

One or more items in the list specified with a NOKEYWORD option were not recognized as reserved words.

* ML 6.0 compatible error

See Also

instruction form requires 80386/486

An instruction was used that was not compatible with the current processor mode.

One of the following processor directives must precede the instruction: .386, .386C, .386P, .486, or .486P.

* ML 6.0 compatible error

See Also

END directive required at end of file

The assembler reached the end of the main source file and did not find an .END directive.

* ML 6.0 compatible error

See Also

too many bits in RECORD: identifier

One of the following occurred:

- Too many bits were defined for the given record field.
- Too many total bits were defined for the given record.
- The size limit for a record or a field in a record is 16 bits when doing 16-bit arithmetic or 32 bits when doing 32-bit arithmetic.

See Also

^{*} ML 6.0 compatible error

positive value expected

A positive value was not found in one of the following situations:

- The starting position specified for SUBSTR or @SubStr
- The number of data objects specified for COMM
- The element size specified for COMM

See Also

^{*} ML 6.0 compatible error

index value past end of string

An index value exceeded the length of the string it referred to when used with INSTR, SUBSTR, @InStr, or @SubStr.

* ML 6.0 compatible error

See Also

count must be positive or zero

The operand specified to the SUBSTR directive, @SubStr macro function, SHL operator, SHR operator, or DUP operator was negative.

* ML 6.0 compatible error

See Also

count value too large

The length argument specified for SUBSTR or @SubStr exceeded the length of the specified string.

* ML 6.0 compatible error

See Also

operand must be relocatable

An operand was not relative to a label. One of the following occurred:

- An operand specified with the END directive was not relative to a label.
- An operand to the SEG operator was not relative to a label.
- The right operand to the minus operator was relative to a label, but the left operand was not.
- The operands to a relational operator were either not both integer constants or not both memory operands. Relational operators can take operands that are both addresses or both non-addresses but not one of each.

* ML 6.0 compatible error

See Also

constant or relocatable label expected

The operand specified must be a constant expression or a memory offset.

* ML 6.0 compatible error

See Also

segment, group, or segment register expected

A segment or group was expected but was not found. One of the following occurred:

- The left operand specified with the segment override operator (:) was not a segment register (CS, DS, SS, ES, FS, or GS), group name, segment name, or segment expression.
- The ASSUME directive was given a segment register without a valid segment address, segment register, group, or the special FLAT group.

See Also

^{*} ML 6.0 compatible error

segment expected: identifier

The GROUP directive was given an identifier that was not a defined segment.

* ML 6.0 compatible error

See Also

invalid operand for OFFSET

The expression following the OFFSET operator must be a memory expression or an immediate expression.

* ML 6.0 compatible error

See Also

segment or group not allowed

An attempt was made to use a segment or group in a way that was not valid. Segments or groups cannot be added.

* ML 6.0 compatible error

See Also

cannot add two relocatable labels

An attempt was made to add two expressions that were both relative to a label.

* ML 6.0 compatible error

See Also

segment exceeds 64K limit

A 16-bit segment exceeded the size limit of 64K.

* ML 6.0 compatible error

See Also

invalid type for data declaration: type

The given type was not valid for a data declaration.

* ML 6.0 compatible error

See Also

HIGH and LOW require immediate operands

The operand specified with either the HIGH or the LOW operator was not an immediate expression.

* ML 6.0 compatible error

See Also

cannot have implicit far jump or call to near label

An attempt was made to make an implicit far jump or call to a near label in another segment.

* ML 6.0 compatible error

See Also

use of register assumed to ERROR

An attempt was made to use a register that had been assumed to ERROR with the ASSUME directive.

* ML 6.0 compatible error

See Also

COMMENT delimiter expected

A delimiter character was not specified for a COMMENT directive.

The delimiter character is specified by the first character that is not white space (spaces or TAB characters) after the COMMENT directive. The comment consists of all text following the delimiter until the end of the line containing the next appearance of the delimiter.

* ML 6.0 compatible error

See Also

conflicting parameter definition

A procedure defined with the PROC directive did not match its prototype as defined with the PROTO directive.

* ML 6.0 compatible error

See Also

PROC and prototype calling conventions conflict

A procedure was defined in a prototype (using the PROTO, EXTERNDEF, or EXTERN directive), but the calling convention did not match the corresponding PROC directive.

* ML 6.0 compatible error

See Also

invalid radix tag

The specified radix was not a number in the range 2-16.

* ML 6.0 compatible error

See Also

INVOKE argument type mismatch: argument number

The type of the arguments passed using the INVOKE directive did not match the type of the parameters in the prototype of the procedure being invoked.

* ML 6.0 compatible error

See Also

language type must be specified

A procedure definition or prototype was not given a language type.

A language type must be declared in each procedure definition or prototype if a default language type is not specified. A default language type is set using either the .MODEL directive, OPTION LANG, or the ASMC command-line options /Gc or /Gd.

* ML 6.0 compatible error

See Also

PROLOGUE must be macro function

The identifier specified with the OPTION PROLOGUE directive was not recognized as a defined macro function.

The user-defined prologue must be a macro function that returns the number of bytes needed for local variables and any extra space needed for the macro function.

* ML 6.0 compatible error

See Also

EPILOGUE must be macro procedure

The identifier specified with the OPTION EPILOGUE directive was not recognized as a defined macro procedure. The user-defined epilogue macro cannot return a value.

* ML 6.0 compatible error

See Also

text macro nesting level too deep

A text macro was nested too deeply. The nesting limit for text macros is 100.

* ML 6.0 compatible error

See Also

missing macro argument

A required argument to @InStr, @SubStr, or a user-defined macro was not specified.

* ML 6.0 compatible error

See Also

VARARG parameter must be last parameter

A parameter other than the last one was given the VARARG attribute.

The :VARARG specification can be applied only to the last parameter in a parameter list for macro and procedure definitions and prototypes. You cannot use multiple :VARARG specifications in a macro.

* ML 6.0 compatible error

See Also

VARARG parameter requires C calling convention

A VARARG parameter was specified in a procedure definition or prototype, but the C, SYSCALL, or STDCALL calling convention was not specified.

* ML 6.0 compatible error

See Also

ORG needs a constant or local offset

The expression specified with the ORG directive was not valid. ORG requires an immediate expression with no reference to an external label or to a label outside the current segment.

* ML 6.0 compatible error

See Also

register value overwritten by INVOKE

A register was passed as an argument to a procedure, but the code generated by INVOKE to pass other arguments destroyed the contents of the register.

The AX, AL, AH, EAX, DX, DL, DH, and EDX registers may be used by the assembler to perform data conversion.

Use a different register.

* ML 6.0 compatible error

See Also

too many arguments to INVOKE

The number of arguments passed using the INVOKE directive exceeded the number of parameters in the prototype for the procedure being invoked.

* ML 6.0 compatible error

See Also

too many operands to instruction

Too many operands were specified with a string control instruction.

* ML 6.0 compatible error

See Also

cannot have more than one .ELSE clause per .IF block

The assembler found more than one .ELSE clause within the current .IF block. Use .ELSEIF for all but the last block.

* ML 6.0 compatible error

See Also

expected data label

The LENGTHOF, SIZEOF, LENGTH, or SIZE operator was applied to a non-data label, or the SIZEOF or SIZE operator was applied to a type.

* ML 6.0 compatible error

See Also

cannot nest procedures

An attempt was made to nest a procedure containing a parameter, local variable, USES clause, or a statement that generated a new segment or group.

* ML 6.0 compatible error

See Also

EXPORT must be FAR: procedure

The given procedure was given EXPORT visibility and NEAR distance. All EXPORT procedures must be FAR. The default visibility may have been set with the OPTION PROC:EXPORT statement or the SMALL or COMPACT memory models.

* ML 6.0 compatible error

See Also

macro label not defined: macrolabel

The given macro label was not found. A macro label is defined with :macrolabel.

* ML 6.0 compatible error

See Also

invalid symbol type in expression: identifier

The given identifier was used in an expression in which it was not valid. For example, a macro procedure name is not allowed in an expression.

* ML 6.0 compatible error

See Also

special register cannot be first operand

A special register was specified to an instruction that cannot take it as the first operand.

* ML 6.0 compatible error

See Also

syntax error in control-flow directive

The condition for a control-flow directive (such as .IF or .WHILE) contained a syntax error.

* ML 6.0 compatible error

See Also

constant value out of range

An invalid value was specified for the PAGE directive. The first parameter of the PAGE directive can be either 0 or a value in the range 10-255. The second parameter of the PAGE directive can be either 0 or a value in the range 60-255.

* ML 6.0 compatible error

See Also

missing right parenthesis

A right parenthesis,), was missing from a macro function call. Be sure that parentheses are in pairs if nested.

* ML 6.0 compatible error

See Also

structure cannot be instanced

An attempt was made to create an instance of a structure when there were no fields or data defined in the structure definition or when ORG was used in the structure definition.

* ML 6.0 compatible error

See Also

structure field expected

The righthand side of a dot operator (.) is not a structure field. This error may occur with some code acceptable to previous versions of the assembler. To enable the old behavior, use OPTION OLDSTRUCTS, which is automatically enabled by OPTION M510 or the /Zm command-line option.

* ML 6.0 compatible error

See Also

unexpected literal found in expression

A literal was found where an expression was expected. One of the following may have occurred:

- A literal was used as an initializer
- A record tag was omitted from a record constant

See Also

^{*} ML 6.0 compatible error

divide by zero in expression

An expression contains a divisor whose value is equal to zero.

Check that the syntax of the expression is correct and that the divisor (whether constant or variable) is correctly initialized.

* ML 6.0 compatible error

See Also

directive must appear inside a macro

A GOTO or EXITM directive was found outside the body of a macro.

* ML 6.0 compatible error

See Also

too few bits in RECORD

There was an attempt to define a record field of 0 bits.

* ML 6.0 compatible error

See Also

invalid qualified type

An identifier was encountered in a qualified type that was not a type, structure, record, union, or prototype.

* ML 6.0 compatible error

See Also

invalid use of FLAT

There was an ambiguous reference to FLAT as a group. This error is generated when there is a reference to FLAT instead of a FLAT subgroup. For example,

```
mov ax, FLAT ; Generates A2178 mov ax, SEG FLAT:_data ; Correct
```

* ML 6.0 compatible error

See Also

structure improperly initialized

There was an error in a structure initializer. One of the following occurred:

- The initializer is not a valid expression.
- The initializer is an invalid DUP statement.

See Also

^{*} ML 6.0 compatible error

initializer must be a string or single item

There was an attempt to initialize a structure element with something other than a single item or string.

This error can be caused by omitting braces ({ }) around an initializer.

* ML 6.0 compatible error

See Also

must use floating point initializer

A variable declared with the REAL4, REAL8, and REAL10 directives must be initialized with a floating-point number or a question mark (?).

This error can be caused by giving an initializer in integer form (such as 18) instead of in floating-point form (18.0).

* ML 6.0 compatible error

See Also

invalid combination with segment alignment

The alignment specified by the ALIGN or EVEN directive was greater than the current segment alignment as specified by the SEGMENT directive.

* ML 6.0 compatible error

See Also

INVOKE requires prototype for procedure

The INVOKE directive must be preceded by a PROTO statement for the procedure being called. When using INVOKE with an address rather than an explicit procedure name, you must precede the address with a pointer to the prototype.

* ML 6.0 compatible error

See Also

symbol language attribute conflict

Two declarations for the same symbol have conflicting language attributes (such as C and PASCAL). The attributes should be identical or compatible.

* ML 6.0 compatible error

See Also

.STARTUP does not work with 32-bit segments

The .STARTUP directive cannot be used in a 32-bit segment; it is valid only when generating 16-bit code.

* ML 6.0 compatible error

See Also

ORG directive not allowed in unions

The ORG directive is not valid inside a UNION definition.

You can use the ORG directive inside STRUCT definitions, but it is meaningless inside a UNION.

* ML 6.0 compatible error

See Also

illegal use of segment register

You cannot use segment overrides for the FS or GS segment registers when generating floating-point emulation instructions with the /FPi command-line option or OPTION EMULATOR.

* ML 6.0 compatible error

See Also

GROUP directive not allowed with /coff option

The GROUP directive is not permitted when compiling with /coff.

See ML and ML64 Command-Line Reference for more information.

* ML 8 compatible error

See Also

must be public or external

An alternate symbol was not public or external.

Example

The code sample below would generate A2217 if the line:

```
public alternate
```

were not present.

```
.686
.model flat

extrn primary(alternate):near
.code

public alternate
alternate:
 ret

end
```

See Also

^{*} ML 8 compatible error

assembly passes reached: passes

* Non ML compatible error

See Also

invalid fixup type for:

* Non ML compatible error

See Also

/PE option requires FLAT memory model

* Non ML compatible error

See Also

/bin: invalid start label

* Non ML compatible error

See Also

cannot use TR-TR with current CPU setting

* Non ML compatible error

See Also

no segment information to create fixup :

* Non ML compatible error

See Also

not supported with current output format:

* Non ML compatible error

See Also

missing .ENDPROLOG:

* Non ML compatible error

See Also

.ENDPROLOG found before EH directives

* Non ML compatible error

See Also

missing FRAME in PROC, no unwind code will be generated

* Non ML compatible error

See Also

size of prolog too big, must be < 256 bytes

* Non ML compatible error

See Also

too many unwind codes in FRAME procedure

* Non ML compatible error

See Also

registers AH-DH may not be used with SPL-DIL or R8-R15

* Non ML compatible error

See Also

multiple overrides

* Non ML compatible error

See Also

unknown fixup type : : at

* Non ML compatible error

See Also

filename parameter must be enclosed in <> or quotes

* Non ML compatible error

See Also

literal expected after '='

* Non ML compatible error

See Also

.SAFESEH argument must be a PROC

* Non ML compatible error

See Also

invalid operand for:

* Non ML compatible error

See Also

invalid fixup type for format: type symbol at location offset

* Non ML compatible error

See Also

cannot open file: filename

Same as ML <u>A1000</u>.

* Non ML compatible error

See Also

I/O error closing file

Same as ML <u>A1001</u>.

* Non ML compatible error

See Also

.CASE redefinition: identifierA(val): identifierB(val)

The given nonredefinable symbol was defined in two places.

* Non ML compatible error

See Also

ASMC Warning A4003

start address on END directive ignored with .STARTUP

Both .STARTUP and a program load address (optional with the END directive) were specified. The address specification with the END directive was ignored.

* ML 6.0 compatible warning

See Also

ASMC Warning A4005

unknown default prologue argument

An unknown argument was passed to the default prologue. The default prologue understands only the FORCEFRAME and LOADDS arguments.

* ML 6.0 compatible warning

See Also

ASMC Warning A4006

too many arguments in macro call

There were more arguments given in the macro call than there were parameters in the macro definition.

* ML 6.0 compatible warning

See Also

ASMC Warning A4007

option untranslated, directive required : option

There is no ASMC command-line equivalent for the given option. The desired behavior can be obtained by using a directive in the source file.

```
Option Directive
/A
/P
/S
.ALPHA
OPTION READONLY
.SEQ
```

* ML 6.0 compatible warning

See Also

invalid command-line option value, default is used: option

The value specified with the given option was not valid. The option was ignored, and the default was assumed.

* ML 6.0 compatible warning

See Also

multiple .MODEL directives found : .MODEL ignored

More than one .MODEL directive was found in the current module. Only the first .MODEL statement is used.

* ML 6.0 compatible warning

See Also

line number information for segment without class 'CODE'

There were instructions in a segment that did not have a class name that ends with "CODE." The assembler did not generate CodeView information for these instructions.

CodeView cannot process modules with code in segments with class names that do not end with "CODE."

* ML 6.0 compatible warning

See Also

cannot open file: filename

The given filename could not be in the current path.

Make sure that filename was copied from the distribution disks and is in the current path.

* ML 6.0 compatible warning

See Also

conditional jump lengthened

A conditional jump was encoded as a reverse conditional jump around a near unconditional jump.

You may be able to rearrange code to avoid the longer form.

* ML 6.0 compatible level 3 warning (-W3)

See Also

procedure argument or local not referenced

You passed a procedure argument or created a variable with the LOCAL directive that was not used in the procedure body.

Unnecessary parameters and locals waste code and stack space. * ML 6.0 compatible level 3 warning (-W3)

See Also

expression condition may be pass-dependent

This warning message may indicate that the code is pass-dependent and should be rewritten.

Example

```
mov al,reg
reg equ <cl>
```

* ML 8.0/Non ML compatible warning

See Also

Invalid command-line option: option

ASMC did not recognize the given parameter as an option. This error is generally caused when there is a syntax error on the command line.

Same as <u>A1006</u>

* Non ML compatible warning

See Also

See <u>A2167</u>

* Non ML compatible warning

See Also

See <u>A2189</u>

* Non ML compatible warning

See Also

See <u>A2103</u>

* Non ML compatible warning

See Also

See <u>A2004</u>

* Non ML compatible warning

See Also

IF[n]DEF expects a plain symbol as argument : argument

* Non ML compatible warning

See Also

instructions and initialized data not supported in segment segments

* Non ML compatible warning

See Also

16bit fixup for 32bit label : *label*

* Non ML compatible warning

See Also

displacement out of range: displacement

* Non ML compatible warning

See Also

no start label defined

* Non ML compatible warning

See Also

no stack defined

* Non ML compatible warning

See Also

for -coff leading underscore required for start label: label

* Non ML compatible warning

See Also

library name is missing

* Non ML compatible warning

See Also

ELF GNU extensions (8/16-bit relocations) used

* Non ML compatible warning

See Also

LOADDS ignored in flat model

* Non ML compatible warning

See Also

directive ignored without -switch switch

* Non ML compatible warning

See Also

ignored: symbol

* Non ML compatible warning

See Also

group definition too large, truncated : group

* Non ML compatible warning

See Also

size not specified, assuming:

This is a warning. may be BYTE, WORD or DWORD. The message may occur if an immediate value is written to an untyped memory reference:

JWasm makes a guess and displays the warning, while ML will display an error in such cases.

* Non ML compatible warning

See Also

ASMC Error Messages | Error A2070

constant expected

* Non ML compatible warning

See Also

ASMC Error Messages | Error A2026

symbol language attribute conflict

Two declarations for the same symbol have conflicting language attributes (such as C and PASCAL). The attributes should be identical or compatible.

Same as ML Error A2192

* Non ML compatible warning

See Also

positive value expected

* Non ML compatible warning

See Also

ASMC Error Messages | Error A2090

register value overwritten by INVOKE

* Non ML compatible warning

See Also

ASMC Error Messages | Error A2133

far call is converted to near call

* Non ML compatible warning

See Also

floating-point initializer ignored

* Non ML compatible warning

See Also

directive ignored:

* Non ML compatible warning

See Also

parameter/local name is reserved word:

* Non ML compatible warning

See Also

.CASE without .ENDC: assumed fall through

In case OPTION SWITCH:C is set (default)

```
.switch eax
  .case 1 : mov al,1 ; warning
  .case 2 : .endc
.endsw
```

See Also

^{*} Non ML compatible warning

cannot delay macro function: <token>

Macro that normally would be delayed expanded.

```
.WHILE macro(<...>)
.ELSEIF macro(<...>)
.CASE macro(<...>)
```

See Also

^{*} Non ML compatible warning

.UNTILAXZ

See <u>.REPEAT</u>

See Also

Directives Reference, .UNTILBXZ, .UNTILCXZ, .UNTILDXZ

.UNTILBXZ

See <u>.REPEAT</u>

See Also

Directives Reference, .UNTILAXZ, .UNTILCXZ, .UNTILDXZ

.UNTILDXZ

See <u>.REPEAT</u>

See Also

Directives Reference, .UNTILAXZ, .UNTILBXZ, .UNTILCXZ

.ENDF

.ENDF closes a .FOR[S] statement.

See Also

<u>Directives Reference</u> | <u>.FOR</u>