

ASM8

A two-pass absolute DOS cross-assembler for the 68HC08/68HCS08

Quick Reference Guide

ASM8 - Copyright © 2001-2005 by Tony Papadimitriou <tonyp@acm.org>
Last Update: September 2005 for ASM8 v1.12

Command-Line Syntax and Options

ASM8 [-option [...]] [[@]filespec [...]] [>errfile]

- *option*(s) may appear before, in between or after *filespec*(s).
- *option*(s) specified apply to all files assembled, regardless of command line placement.
- Text file(s) containing list(s) of files to be processed may be specified by naming the text file on the command line, prefixed with a «@» character. These text files may not contain command line options.
- *filespec*(s) may include wildcard characters (?,*). Wildcards are not allowed in *filespec*(s) that are prefixed with a «@» but are allowed in filespecs inside @files.
- If the file extension for a source *filespec* is omitted, the extension «.ASM» is assumed (see description of the -R.ext option below).
- Assembler errors may be redirected to *errfile* using standard DOS output redirection syntax. This capability may be used in conjunction with, or as an alternative to the -E+ option.
- The assembler will set the DOS ERRORLEVEL variable when it terminates as indicated:
 - 0 No error, assembly of last file was successful
 - 1 System error (hardware I/O failure, out of disk space, etc.), or -w option failure
 - 2 Error(s) generated (or Escape pressed) during assembly of last file
 - 3 Warning(s) generated during assembly of last file
 - 4 Assembler was not started (help screen displayed, -w option used with success)

Option	Default	Description
-C[±]	-C-	Label case sensitivity: + = case sensitive (See also #CASEON/#CASEOFF)
-Dlabel [:expr]		Use up to ten times to define symbols for use with conditional assembly (IFDEF and IFNDEF directives). Symbols are always uppercase (regardless of -C option). If they are not followed by a value (or expression) they assume the value zero. Expression is limited to 19 characters. Character constants should not contain spaces, and they are converted to uppercase. Cannot be saved with -w.
-E[±]	-E-	Generate *.ERR file (one for each file assembled). *.ERR files are not generated for file(s) that do not contain errors.
-EH[±]	-EH+	If -E+ is in effect, hide (do not display) error messages on screen.
-EXP[±]	-EXP-	When on, a .EXP file is created containing all symbols defined with an EXP rather than an EQU pseudo-opcode. The resulting file can be used as an #INCLUDE file for other programs. This allows for automatic creation of include files with global exported symbols.
-HCS[±]	-HCS-	When on, the assembler understands the extended HCS08 instruction set. The cycle counts in the listing also reflect the HCS08 core. To check the current status of this switch look at the help screen's second line from top.

		The software will say it's either a MC68HC08 or a MC68HCS08 assembler based on this setting. See also the directives #HCSON, #HCSOFF, #IFHCS, and #IFNHCS
-Ix		Define default INCLUDE directory root. Relative path files not found relative to the main file, will be tried next relative to this directory. Absolute path file definitions are not affected by this switch. Affects both the INCLUDE and the IF(N) EXISTS directives.
-L[±]	-L+	Create a *.LST file (one for each file assembled).
-LC[±]	-LC+	List any conditional directives fully (the directives only, not the contents in between), even when they are False.
-LS[±]	-LS-	Create a *.SYM symbol list (one for each file assembled). May be useful for debuggers that do not support the P&E map file format.
-LSx	-LSS	x may be either S (default) for simple SYM file, E for EM11/Shadow11 SYM compatible format (possibly not useful for HC08), or N for NoICE SYM format.
-M[±]	-M+	Create a *.MAP (one for each file assembled). *.MAP files created may be used with debuggers that support the P&E source-level map file format.
-MTx	-MTP	Specifies type of MAP file to be generated (if -M+ in effect): -MTA : Generate parsable ASCII map file -MTP : Generate P&E-style map file
-O[±]	-O+	Enables two warnings: S19 overlap, and Violation of MEMORY directive.
-P[±]	-P+	When on it tells the assembler to stop after Pass 1 if there were any errors. Provides for faster overall assembly process and less confusion by irrelevant side errors of Pass 2. Warnings do not affect this.
-Q[±]	-Q-	Specifies quiet run (no output) when redirecting to an error file. Useful for IDEs that call ASM8 and don't want to have their display messed up.
-Rn	-R74	Specifies maximum length of S-record files. The length count n includes all characters in an S-record, including the leading «S» and record type, but not the CR/LF line terminator. Minimum value is 12 while maximum is 250.
-R.ext	-R.ASM	Specifies the default extension to assume for source files specified on the command line which do not directly specify an extension.
-REL[±]	-REL+	Allows generation of «BRA/BSR instead of JMP/JSR» optimization warnings when enabled. (See also OPTRELON/OPTRELOFF)
-RTS[±]	-RTS-	Allows generation of «JSR followed by RTS» subroutine call optimization warnings when enabled. (See also OPTRTSON/OPTRTSOFF)
-S[±]	-S+	Generate *.S19 object file (one for each file assembled).
-SH[±]	-SH-	Include dummy «S0» record (header) in object file (only if -s+).

-SP [±]	-SP-	When enabled, the operand part of an instruction is stripped of spaces before parsing. In this case, possible comments must begin with semi-colon. (See also SPACESON/SPACESOFF)
-T [±]	-T-	Makes redirected errors look like Turbo Pascal errors (useful to fool certain IDEs).
-T <i>n</i>	-T8	Specifies tab field width to use in *.LST files. Tab characters embedded in the source file are converted to spaces in the listing file such that columns are aligned to 1 + every <i>n</i> th character.
-X [±]	-X+	Allow recognition of extra, non-68HC08-standard mnemonics in source files. (See also EXTRAON/EXTRAOFF)
-WRN [±]	-WRN+	Enables or disables the display of all warnings. When enabled, only warnings that aren't disabled individually will be generated. When disabled, it overrides local warning options (such as -REL and -RTS).
-W	(none)	Write options specified on command line to the ASM8 executable. The user-specified options become the default values used by ASM8 in subsequent invocations. <i>Filespec(s)</i> on the command line are ignored. Assembly of source files does not take place if this option is specified.

Source File Pseudo-Opcodes

Pseudo-Op	Description
DB <i>string</i> <i>expr</i> [,...]	Define Byte(s). <i>expr</i> may be a constant numeric, a label reference, an expression, or a string. DB encodes a single byte in the object file at the current location counter for each <i>expr</i> encountered (using the LSB of the result) or one byte for each character in <i>strings</i> .
DS <i>blocksize</i>	Define Storage. The assembler's location counter is incremented by <i>blocksize</i> . Forward references not allowed. No code is generated.
DW <i>expr</i> [,...]	Define Word(s). <i>expr</i> may be a constant numeric, a label or an expression. <i>expr</i> is always interpreted as a word (16-bit) quantity, and is stored in the object file at the current location counter, high byte followed by low byte.
END [<i>expr</i>]	Provided for compatibility. The END directive cannot be used to terminate assembly; ASM8 always processes the source file to the end of file. If <i>expr</i> is specified, the word result is encoded in the S9 record of the object file.
<i>label</i> EQU <i>expr</i>	Assigns the value of <i>expr</i> to <i>label</i> . See also EXP
<i>label</i> EXP <i>expr</i>	Assigns the value of <i>expr</i> to <i>label</i> . This is identical to EQU but with the following difference: Labels defined thus will be included in the .EXP file as regular EQU s. This effectively allows to export symbols for use from other source files. It makes it possible to give only object code to others along with the produced .EXP file so that they can «link» the object to their source.
FCB <i>string</i> <i>expr</i> [,...]	Form Constant Byte(s). Same as DB .
FCC <i>string</i> <i>expr</i> [,...]	Form Constant Character(s). Same as DB .
FCS <i>string</i> <i>expr</i> [,...]	Form Constant String. Similar to FCC , but automatically adds a terminating null (0) byte to the end of the string defined (for ASCII strings).
FDB <i>expr</i> [,...]	Form Double Byte(s). Same as DW .
ORG <i>expr</i>	Sets the assembler's location counter for the active segment. Code generated after this directive will be assembled starting at the location specified by <i>expr</i> .
RMB <i>blocksize</i>	Reserve Memory Byte(s). Same as DS .

Source File Processing Directives

- All processing directives must be prefixed with a \$ or # character. ASM8 will recognize either character as the start of a processing directive.
- If a directive has a corresponding command-line option, the directive in the source file will override the command line directive at the point in which the source file directive is encountered.
- [*text*] will be trimmed of duplicate spaces. To have more than one consecutive spaces display, use the Alt-255 character, as many times as needed.

Directive	Description
#CASEOFF	When #CASEOFF is in effect, all symbol references that follow are converted to uppercase internally before they are searched for or placed in the symbol table. Equivalent to the -c- command line option.
#CASEON	When #CASEON is in effect, symbol references are NOT internally converted to uppercase before they are searched for or placed in the symbol table. Equivalent to the -c+ command line option.
#DATA	Activation of the DATA segment. Default starting value is \$FE00.
#EEPROM	Activation of the EEPROM segment. Default starting value is \$0000.
#EJECT	See #PAGE
#ELSE	When used in conjunction with conditional assembly directives (#IF , #IF[N]DEF , \$IF[N]Z , #IFMAIN , #IFINCLUDED , etc.), code following the #ELSE directive is assembled if the conditional it is paired with evaluates to a not-true result.
#ENDIF	Marks the end of a conditional-assembly block. Conditional assembly statements may be nested if they are properly blocked with #ENDIF directives.
#ERROR [<i>text</i>]	When encountered in the source, the assembler issues a error message in the same form as internally-generated errors, using the <i>text</i> specified, prefixed with «USER: »
#EXTRAOFF	Disables recognition of ASM8's extended instruction set for source lines that follow this directive. Equivalent to the -x- command line option.
#EXTRAON	Enables recognition of ASM8's extended instruction set for source lines that follow this directive. Equivalent to the -x+ command line option.
#FATAL [<i>text</i>]	Similar to the #ERROR directive, but generates an assembler fatal error message and terminates the assembler (possible further files in the list will not be processed).

#HCSOFF	Disables the HCS08 instruction set mode. See also #IFHCS #IFNHCS #HCSON Equivalent to the -HCS- command line option.
#HCSON	Enables the HCS08 instruction set mode. See also #IFHCS #IFNHCS #HCSOFF Equivalent to the -HCS+ command line option.
#IF <i>expr1 cond expr2</i>	Evaluates <i>expr1</i> and <i>expr2</i> (which may be any valid ASM8 expression) and compares them using the specified <i>cond</i> conditional operator. If the condition is true, the code following the #IF operator is assembled, up to its matching #ELSE or #ENDIF directive. <i>Cond</i> may be any one of: < <= = >= > <> The condition is always evaluated using unsigned arithmetic. If a symbol referenced in <i>expr1</i> or <i>expr2</i> is not defined, the statement will always evaluate as false.
#IFDEF <i>expr</i>	Attempts to evaluate <i>expr</i> , and if successful, assembles the code that follows, up to the matching #ELSE or #ENDIF directive. This directive is usually used to test if a specified symbol has been defined. Symbol(s) referenced in <i>expr</i> must be defined before the directive for the result to evaluate true (e.g., forward references will evaluate as false). #IFDEF without an <i>expr</i> following will always evaluate to False.
#IFEXISTS <i>fpath</i>	Checks for the existence of the file specified by <i>fpath</i> (using the same rules as those used for #INCLUDE directives) and assembles the code which follows if the specified <i>fpath</i> exists.
#IFHCS	Assembles the following code if the assembler is in the extended HCS08 instruction set mode. See also #IFNHCS #HCSON #HCSOFF
#IFINCLUDED	Assembles the code which follows if the file containing this directive is a file used in an INCLUDE directive of a higher-level file (regardless of nesting level). See also #IFMAIN
#IFMAIN	Assembles the code which follows if the file containing this directive is the main (primary) file being assembled. See also #IFINCLUDED .
#INCLUDE <i>fpath</i>	INCLUDES the specified <i>fpath</i> file in the assembly stream, as if the contents of the file were physically present in the source at the point where the #INCLUDE directive is encountered. INCLUDES may be nested, up to 100 levels (the main source file counts as one level). Relative <i>fpath</i> specifications are always referenced to the directory in which the main source file resides, including any relative #INCLUDE <i>fpath</i> references in nested include files. Note: ASM8 will only generate a standard error (not an assembly-

	terminating fatal error) if a file specified in a #INCLUDE directive is not found. The #IFEXISTS and #IFNEXISTS directives may be used in conjunction with #FATAL if termination of assembly is desired under such conditions.
#IFNDEF <i>expr</i>	Evaluates <i>expr</i> and assembles the code that follows if the expression could NOT be evaluated; usually as the result of a reference to an undefined symbol. This directive is the functional opposite of the #IFDEF directive.
#IFNEXISTS <i>fpath</i>	The opposite of #IFEXISTS ; code following this directive is assembled if the specified <i>fpath</i> does NOT exist. The -lx directory will be used also to determine if a file exists or not.
#IFNHCS	Assembles the following code if the assembler is in the regular HC08 instruction set mode. See also #IFHCS #HCSON #HCSOFF
#IFNZ <i>expr</i>	Evaluates <i>expr</i> and assembles the code that follows if the expression evaluates to a non-zero value. #IFNZ always evaluates to false if <i>expr</i> references undefined or forward-defined symbols.
#IFZ <i>expr</i>	Evaluates <i>expr</i> and assembles the code that follows if the expression is equal to zero. #IFZ always evaluates to false if <i>expr</i> references undefined or forward-defined symbols.
#LIST	See #LISTON
#LISTOFF	Turns off generation of source and object data in the *.LST file for all lines which follow this directive. Useful for excluding the contents of #INCLUDE files in the *.LST file.
#LISTON	Enables generation of source and object data in the *.LST file for the source code following this directive. Has no effect if list file generation is disabled (-L- command line option in effect).
#MAPOFF	Suppresses generation of source-line information in the *.MAP file for the code following this directive. Symbols which are defined following this directive are still included in the *.MAP file.
#MAPON	Enables generation of source-line information in the *.MAP file for the code following this directive. #MAPON is the default state when assembly is started when map file generation is enabled (-M+ command line option).
#MEMORY <i>addr1</i> [<i>addr2</i>]	Maps a memory location (or range, if <i>addr2</i> is also supplied) of code and/or data areas as valid. Use multiple directives to specify additional ranges. Any code or data that falls outside the given range(s) will produce a warning (if the -o option is enabled) for each violating byte. Very useful for segmented memory devices, etc. <i>Addr1</i> and <i>addr2</i> may be specified in any order. The range defined will always be between the smaller and the higher values.
#MESSAGE [<i>text</i>]	Displays <i>text</i> on screen during the first pass of assembly when this directive is encountered in the source. Messages are not

	written to the error file. They are meant to inform the user of the options used or conditional path taken.
#NOLIST	See #LISTOFF
#OPTRELOFF	Disable «BRA/BSR instead of JMP/JSR» optimization warnings. Equivalent to the -REL- command line option.
#OPTRELON	Enable warning generation when an absolute branch or subroutine call (JMP or JSR) is encountered that could be successfully implemented using the relative form of the same instruction (BRA or BSR). This option is on by default. Equivalent to the -REL+ command line option.
#OPTRTSOFF	Disable RTS-after-JSR/BSR optimization warning (default). Equivalent to the -RTS- command line option.
#OPTRTSON	Enable warning generation when a subroutine call (JSR or BSR) is immediately followed by a RTS. This option is off by default. Command-line option -RTS+ does the same thing.
#S19FLUSH	Forces the immediate termination of an S-record line when encountered, rather than waiting for the record to reach the size specified by the -Rn command line directive. This directive may be used to make identification of the end of code blocks easier when viewing the *.S19 file.

#PAGE	Outputs a Form Feed (ASCII 12) character followed by a Carriage Return (ASCII 13) in the *.LST file just before displaying the line that contains this directive.
#PUSH	Pushes on an internal stack the current settings of the following options: MAPx , LISTx , CASEx , EXTRAx , SPACESx , OPTRELx , OPTRTSx , HCSx , and TABSIZE . Useful in included files that want to change any of these options without affecting parent files. See also #PULL
#PULL	Pulls from an internal stack the last pushed settings of the following options: MAPx , LISTx , CASEx , EXTRAx , SPACESx , OPTRELx , OPTRTSx , HCSx , and TABSIZE . Useful in included files that want to change any of these options without affecting parent files. You can use PULL even if haven't used PUSH , no action will take place. See also #PUSH
#RAM	Activation of the RAM segment. Default starting value is \$0080.
#ROM	Activation of the ROM segment. Default starting value is \$F600. This is the default segment if none is specified.
#SEG_n	Activation of the SEG _n segment (n is a number from 0 through 9). Default starting value for all ten segments is \$0000.
#TABSIZE <i>n</i>	Specifies the field width of tab stops used in the source file. Proper use of this directive ensures that the *.LST files generated by ASM8 are formatted in the same way as your source files appear in your text editor. This directive overrides the setting of the -Tn command line option for the source file(s) in which it is encountered.
#VECTORS	Activation of the VECTORS segment. Default starting value is \$FFC0.
#WARNING [<i>text</i>]	Similar to the #ERROR directive, but generates an assembler warning message instead of an error message.
#XRAM	Activation of the XRAM segment. Default starting value is \$2000.
#XROM	Activation of the XROM segment. Default starting value is \$EC00.

Expression Operators and Other Special Characters Recognized by ASM8

- Expressions are evaluated in the order they are written (left to right).
All operators have equal precedence.
- Avoid inserting spaces between values and operators (unless using -SP+ switch and ; comments).

Operator	Description
+	Addition
-	Subtraction When used as a unary operator, the 2's complement of the value to the right is returned.
*	Multiplication Can also be used to represent the current location counter.
/	Integer Division (ignores remainder)
\	Modulus (remainder of integer division)
=	'Equal' comparison for the \$IF directive.
<>	'Not equal' comparison for the \$IF directive.
>=	'Greater than or equal' comparison for the \$IF directive.
>	Shift right – operand to the left is shifted right by the count to the right. Also used to specify extended addressing mode. 'Greater than' comparison for the \$IF directive.
<=	'Less than or equal' comparison for the \$IF directive.
<	Shift left – operand to the left is shifted left by the count to the right. Also used to specify direct addressing mode. 'Less than' comparison for the \$IF directive.
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR (exclusive OR)
~	Swap high and low bytes (unary): ~\$1234 = \$3412 Useful for converting word values from big-endian to little-endian or the inverse.
[Extract low 8 bits (unary): [\$1234 = \$34
]	Extract high 8 bits (unary):]\$1234 = \$12
\$	Interpret numeric constant that follows as a hexadecimal number. Can also be used to represent the current location counter.
%	Interpret numeric constant that follows as a binary number
' - "	Any one of these characters (single, back, or double-quote) may be used to enclose a string or character entity. The character used at the start of the string must be used to end it.
#	Specifies immediate addressing mode
@	Specifies direct addressing mode (same as «<»)

ASM8 Extended Instruction Set

The instructions listed below are not actually new instructions, rather, internal macros that generate one or more 68HC08/68HCS08 CPU instructions. These instructions are only recognized if the extended instruction set option is enabled (-x+ command line option or #EXTRAON processing directive).

Mnemonic/Syntax	Description
AAX	Add A to H:X Same as: PSHA/PSHX/ADD 1,SP/PULX/TAX/BCC ?/PSHH/INC 1,SP/PULH/? PULA
ADDHX #wordval	Add immediate value to HX (useful for table offset adjustment) Same as: PSHA/TXA/ADD #LO/TAX/PSHH/PULA/ADC #HIGH/PSHA/PULH/PULA
CLRHX	Same as: CLRH/CLRX
CMPA operand	Same as: CMP operand
CMPX operand	Same as: CPX operand
JCC addr16	Jump equivalent to BCC (BCS \$+5 followed by JMP addr16)
JCS addr16	Jump equivalent to BCS (BCC \$+5 followed by JMP addr16)
JEQ addr16	Jump equivalent to BEQ (BNE \$+5 followed by JMP addr16)
JGE addr16	Jump equivalent to BGE (BLT \$+5 followed by JMP addr16)
JGT addr16	Jump equivalent to BGT (BLE \$+5 followed by JMP addr16)
JHCC addr16	Jump equivalent to BHCC (BHCS \$+5 followed by JMP addr16)
JHCS addr16	Jump equivalent to BHCS (BHCC \$+5 followed by JMP addr16)
JHI addr16	Jump equivalent to BHI (BLS \$+5 followed by JMP addr16)
JHS addr16	Jump equivalent to BHS (BLO \$+5 followed by JMP addr16)
JIH addr16	Jump equivalent to BIH (BIL \$+5 followed by JMP addr16)
JIL addr16	Jump equivalent to BIL (BIH \$+5 followed by JMP addr16)
JLE addr16	Jump equivalent to BLE (BGT \$+5 followed by JMP addr16)
JLO addr16	Jump equivalent to BLO (BHS \$+5 followed by JMP addr16)
JLS addr16	Jump equivalent to BLS (BHI \$+5 followed by JMP addr16)
JLT addr16	Jump equivalent to BLT (BGE \$+5 followed by JMP addr16)
JMC addr16	Jump equivalent to BMC (BMS \$+5 followed by JMP addr16)
JMI addr16	Jump equivalent to BMI (BPL \$+5 followed by JMP addr16)
JMS addr16	Jump equivalent to BMS (BMC \$+5 followed by JMP addr16)
JNE addr16	Jump equivalent to BNE (BEQ \$+5 followed by JMP addr16)
JPL addr16	Jump equivalent to BPL (BMI \$+5 followed by JMP addr16)
LSLHX	Logical shift left of HX (useful for table offset adjustment) Same as: PSHH/LSLX/ROL 1,SP/PULH
OS byteval	Operating system call: SWI / DB byteval

OSW <i>wordval</i>	Operating system call:	SWI / DW <i>wordval</i>
PSHHX	Same as:	PSHX / PSHH
PULHX	Same as:	PULH / PULX
TAH	Same as:	PSHA / PULH
THA	Same as:	PSHH / PULA
THX	Same as:	PSHH / PULX
TXH	Same as:	PSHX / PULH
INX	Same as:	INCX
DEX	Same as:	DECX
XGAX	Same as:	PSHA / TXA / PULX

ASM8-generated Error and Warning Messages

This section provides the lists of error and warning messages.

Errors inform the user about problems that prevent the assembler from producing 'correct' code. If there is even a single error during assembly, no files will be created (except for the ERR file, if one was requested).

Warnings inform the user about problems that do not prevent the assembler from producing 'correct' code but the code produced may not be what was intended, or it may be inefficient. A program that has warnings may be totally correct and run as expected.

Errors and warnings that begin with 'USER:' are generated by #ERROR and #WARNING directives, respectively. The source code author decides their meaning and importance.

In the lists below, what's enclosed in angle brackets (< and >) is a 'variable' part of the message. That is, it is different depending on the source line to which the error or warning refers.

The order the messages appear below is random. Some messages have similar meanings; they simply result from different checks of the assembler.

ERRORS

1. Invalid binary number

The string following the % sign is not made up of zeros and/or ones.

2. Binary number is longer than 16 bits

A binary number may have no more than sixteen significant digits. Leading zeros are ignored.

3. "<SYMBOL>" not yet defined, forward refs not allowed

RMB and DS directives may not refer to forward defined symbols. You must define the symbol(s) used in advance.

4. Bad <MODE> instruction/operand "<OPCODE> <OPERAND>

The instruction and operand addressing mode combination is not a valid one, or you have turned the -X option (EXTRAx directive) off. For example, TST #4 will show «*Bad IMMEDIATE instruction/operand "TST 4"*» because although TST is a valid instruction, it does not have an immediate addressing mode option.

5. Could not close MAP file

For some reason, the MAP file could not be closed. Possibly some disk problems (check available space, etc.) If you can't figure out what's wrong and still must assemble, turn off the MAP with the -M- option.

6. Could not close SYM file

For some reason, the SYM file could not be closed. Possibly some disk problems (check available space, etc.) If you can't figure out what's wrong and still must assemble, turn off the SYM with the -S- option.

7. Could not create MAP file <FILEPATH>

For some reason, the MAP file could not be created. Possibly some disk problems (check available space, etc.) If a MAP file of the same name already exists it probably has a read-only attribute or is somehow locked by the system.

8. Could not create SYM file <FILEPATH>

For some reason, the SYM file could not be created. Possibly some disk problems (check available space, etc.) If a SYM file of the same name already exists it probably has a read-only attribute or is somehow locked by the system.

9. Expression error

Something is wrong with the attempted expression, or an expression is altogether missing.

10. Invalid argument for DB directive

The value or expression supplied is not correct.

11. Invalid argument for EQU/EXP directive

The value or expression supplied is not correct.

12. Invalid first argument

The first value or expression supplied is not correct.

13. Invalid second argument

The second value or expression supplied is not correct.

14. Missing value between commas

Two (or more) commas without a value in between.

15. Possibly duplicate symbol "<SYMBOL>"

The symbol shown has already been defined. The word 'possibly' suggests that a symbol may have been truncated to 19 characters, and thus not appear duplicate to the user, only to the assembler. It also suggests that the original may have been written for case-sensitive assembly but you turned the option off.

16.Repeater value is invalid

The repeater value (the `:n` part of the opcode) is not a positive integer number.

17.Symbol "<SYMBOL>" contains invalid character(s)

The symbol shown contains characters that are used in special ways and, therefore, cannot be part of a symbol because they will cause ambiguities. For example, a quote within a symbol is not allowed.

18.Undefined symbol "<SYMBOL>" or bad number

The string shown is either a symbol that hasn't been defined at all, or it is a number that has some error, for example: `$ABCH` and `$FFFFFF` are not valid hex numbers. The first contains an invalid character while the second is greater than 16 significant bits (`$FFFF`).

19.USER: <USER TEXT>

This is a user generated error via the `#ERROR` directive.

20.Comma not expected

A comma was found in an unexpected position within the operand. Possibly using more arguments than required.

21.Syntax error

Some symbol is confusing the assembler. For example, `FCB # $FF` will give a syntax error because the `#` indicates immediate addressing mode which makes no sense for an `FCB` directive (the correct is `FCB $FF`).

22.Empty string not allowed

An empty string (two quotes next to each other) is not allowed because there is no value that can be generated from it.

23.Could not open include file <FILEPATH>

The `[path and]` file shown could not be located or opened. If the file exists, it may be locked by some other program (under Windows, the file could be loaded in an editor).

24.ELSE without previous *ifxxx*

An `#ELSE` directive was encountered that does not match any unmatched `#IF` directive.

25.ENDIF without previous *ifxxx*

An `#ENDIF` directive was encountered that does not match any unmatched `#IF` directive.

26.Forward references not allowed

The `#MEMORY` and other directives do not accept forward references.

27.Incomplete argument for Bit Instruction (commas?)

`BSET`, `BCLR`, `BRSET`, and `BRCLR` require commas between each part of the operand. You have either left the commas out or forgotten to supply all the parts of the operand.

28.Invalid expression(s) and/or comparator

The expression or comparator used in the `#IF` directive is incorrect.

29.Missing branch address

`BRSET` and `BRCLR` require a target address for branching to but one was not supplied. The branch target is the last part of the operand and it can be any valid expression.

30.Missing INCLUDE filename

An `#INCLUDE` directive was supplied without any `[path and]` filename.

31. Missing required first address

A #MEMORY directive was encountered without any value or expression.

32. Repeater value out of range (1-32767)

The repeater value (the :n part of the opcode) must be from 1 to 32767.

33. Required string delimiter not found

You have supplied only one quote to a string, or the string is inappropriately separated from the previous or next operand. For example: 'ABC' CR lacks a comma between the quote and the CR symbol. So, the found string ACB' CR is invalid.

34. Symbol "<SYMBOL>" does not start with A..Z, . or _

All symbols must start with one of the above characters. (Local symbols start with a ?)

35. Symbol "<SYMBOL>" is reserved for indexing modes

You have used a symbol named X or Y. These names are not allowed because they cause ambiguities with the X and Y registers in the various indexed mode instructions.

36. Too many include files. Maximum allowed is 99

The maximum number of INCLUDE files is 99 (regardless of nesting level). You have gone over this number. Possible solution: Combine related files together as required. Keep in mind that although the assembler allows this many files to be included, a lot of programs cannot handle these many files in the MAP files.

37. Division by zero

The expression used contains a division by zero after the / operator.

38. MOD division by zero

The expression used contains a division by zero after the \ operator.

39. "<SYMBOL>" is too far back [<VALUE>], use jumps

The target of a branch instruction is too far back by as many bytes as shown. Either get it closer to the target, or use a jump instead. Some instructions, like BRCLR or BRSET do not have an equivalent jump so you must use an intermediate 'jump hook' instead.

40. "<SYMBOL>" is too far forward [<VALUE>], use jumps

The target of a branch instruction is too far forward by as many bytes as shown. Either get it closer to the target, or use a jump instead. Some instructions, like BRCLR or BRSET do not have an equivalent jump so you must use an intermediate 'jump hook' instead.

41. Invalid argument for DW or FDB directive

The value or expression supplied is not correct.

42. Invalid argument for ORG directive

The value or expression supplied is not correct.

43. Invalid argument for RMB or DS directive

The value or expression supplied is not correct.

44. Invalid argument for END directive

The value or expression supplied is not correct.

WARNINGS

1. Direct mode wasn't used (forward reference?)

Automatic Direct Mode detection requires that symbol(s) used be defined in advance. You should either define the referenced symbol(s) earlier in your code, or use the Direct Mode Override (<) to force the assembler to use Direct Addressing Mode.

2. Label on left side of END line ignored

The `END` directive does not take a label. If one is used it will be ignored (it will not be defined).

3. Label on left side of ORG line ignored

The `ORG` directive does not take a label. If one is used it will be ignored (it will not be defined).

4. Trailing comma ignored

A multiple-parameter pseudo-instruction was used (such as `FCB` and `DW`) and a comma was found at the end of the parameter list. This may indicate the list is separated by both commas and spaces. You must either remove the spaces or assemble with `#SPACESON` (or the `-SP+` option).

5. Violation of MEMORY directive at address \$<VALUE>

ASM8 has produced code and/or data that falls outside any address ranges defined via the `#MEMORY` directive. You must either add more `#MEMORY` directives to cover the offending range or move your code/data elsewhere (using appropriate segment and/or `ORG` statements).

6. EQU/EXPs require a label, ignoring line

A `EQU` (or `EXP`) by definition is meant to assign a value to a symbol but no symbol name was supplied. Using a repeater value in an `EQU` will also produce this warning for each repetition of the statement except the first one. You should NOT use repeaters with `EQU`.

7. Forward references are always FALSE

Conditional directives other than `#IFDEF` and `#IFNDEF` produce this warning if the symbol(s) referenced have not yet been defined. In this case, the conditional evaluates to false, and if there is an `#ELSE` part, it is taken.

8. String is too long, only first 8 used

8-bit instructions (such as `LDA`) cannot accept a constant string value of more than 8 bits. The longer string encountered is truncated before being used.

9. S19 overlap at address \$<VALUE>

The code/data of the shown line overlaps an already occupied memory location at the address shown. The warning appears at code/data that causes the first and consequent overlaps but the problem could be with the original code/data that occupied this address. The assembler has no way of knowing your intentions!

10. Extra operand found ignored

In a `BCLR/BSET` you have supplied a branch address. Depending on what you intended to do, either change the instruction to `BRCLR/BRSET` or remove the last operand.

11. No ending string delimiter found

The last string quote is missing. ASM8 did its best to produce a value for you but it may not be the one you wanted. For example: `«LDA #'a»` will produce this warning but the value used will be correct, while `«LDA #'a ;comment»` will produce a wrong value (the space after the `a` because the string `'a'` is a 16-bit value downsize to an 8-bit value, you will get a warning about this also).

12. Operand is larger than 16 bits, using low 16-bits

The operand is greater than 16 bits but the instruction can only accept a 16-bit operand. The lower word was used.

13.Operand is larger than 8 bits, using low 8-bits

The operand is greater than 8 bits but the instruction can only accept a 8-bit operand. The lower byte was used.

14.Possible memory wraparound at address \$<VALUE> (<DEC VALUE>)

It seems like you have reached the end of memory (\$FFFF) and caused the Program Counter to wrap around to zero. In some situations this may be intentional. Using `RMB 2` (rather than `FDB` or `DW`) in the vector for RESET will also give this warning but it should be ignored. The address shown is the beginning address of the [pseudo-] instruction that caused the wraparound.

15.A JUMP was used when a BRANCH would also work

You could have used a Branch instead of a Jump. This will make your code one byte shorter for each warning. Controlled by the `-REL (OPTRELxx)` option.

16.Attempting operation with missing first operand

An operation was attempted without an operand before the operator. For example `/3` (divide by 3) is missing the dividend.

17.JSR/BSR followed by unlabeled RTS => JMP/BRA

You could safely replace the sequence `JSR/RTS` or `BSR/RTS` to a single `JMP` or `BRA`, accordingly. The code will remain equivalent but you will gain a byte of memory, two bytes of stack space, and also make it a little faster. It will, however, make your source-code less user-friendly and a bit harder to follow. It should probably be done only when speed is very important or if you're running out of space and must save every byte you can. **WARNING:** In certain situations, the code is dependent on the return address pushed on the stack by a `JSR` or `BSR` instruction. In those cases, do NOT replace with `JMP/BRA` because the code will not run correctly. It is assumed you know the code you're working on. Controlled by the `-RTS (OPTRTSxx)` option.

18.No ORG (RAM:\$0080 ROM:\$F600 DATA:\$FE00 VECTORS:\$FFDE)

ASM8 started producing code/data without having been told explicitly where to put it. A segment directive may have been used, however, with its default value. **NOTE:** You will only get this warning once no matter how many segments you are using. This means that you may be required to add `ORGs` for each segment or else the default values will be used.

19.Phasing on <SYMBOL> (PASS1: \$<VALUE>, PASS2: \$<VALUE>')

The symbol shown was defined two or more times using different values. The values given may help you determine the type of the problem more quickly. If it's a duplicate label with the same purpose or a completely random use of the same symbol name. The assembler will attempt to use the last (most current) value for this symbol.

20.TABSIZE must be a positive integer number, not changed

The `TABSIZE` directive requires a positive integer, and one wasn't supplied. The current tab size was not altered.

21.Unrecognized directive "<DIRECTIVE>" ignored

Something that looks like a directive (i.e., begins with `#` or `$` and appears first in a line after the white-space) was encountered but it wasn't a valid one. Check spelling. If spelling seems correct, you may be assembling someone else's code written for a later version of ASM8 that supports additional directives your version doesn't understand.

22.USER: <USER TEXT>

This is a user generated warning via the #WARNING directive

23. Branching to next instruction is needless

You are using a branch instruction (other than BSR) to send control to the immediately following instruction. This is the default action of the CPU, so this instruction is not required. Controlled by the -REL (OPTRELxx) option.

24. Jumping to next instruction is needless

You are using a Jxx instruction (other than JSR) to send control to the immediately following instruction. This is the default action of the CPU, so this instruction is not required. Controlled by the -REL (OPTRELxx) option.

25. Instruction BGND is only valid in BDM

This instruction is practically never used in a user program (except perhaps to force an illegal opcode exception). It's only good for debugging purposes in the special BGND mode of HCS08 CPUs only. Not applicable to the HC08 CPU. Available only when -HCS+ (HCSON).

ASM8's Miscellaneous Features

Repeaters

Each opcode or pseudo-opcode may be suffixed by a colon [:] and a positive integer between 1 and 32767. This is referred to as the <repeater value>. The repeater value must be provided explicitly, no expressions or symbols are allowed, just a plain number. Some examples:

LSRA:4		;Move high nibble to low
FCB:256	0	;Create a table of 256 zeros

Segments

Eight special directives allow you to use segments in your programs. Segments are useful mostly in conjunction with the use of `INCLUDE` files. Since often it is not possible to know the current memory allocation for variables and code when inside a general-purpose `INCLUDE` file, segments help overcome this (and other problems) with ease.

Also, we often want to have our code and data (strings, tables, etc.) grouped in a different way in our source-code than the resulting S19 (object). Segments again give us the ability to have related code, variables, and data together in the source but separated into distinct memory areas in the object file. When using segments, it is common to have a single `ORG` statement for each of the segments, near the beginning of the program. Thereafter, each time we need to «jump» to a different memory segment/area, we use the relevant segment directive.

Although the eight segments are named `#RAM`, `#ROM`, `#EEPROM`, `#XRAM`, `#XROM`, `#DATA`, `#SEGn`, and `#VECTORS` their use is identical (except for the initial default values) and they are interchangeable. Use of segments is optional. If segments aren't used, you are always in the default `#ROM` segment (which explains why code assembles beginning at `$F600`).

Local Symbols

All symbols that begin with a question mark [?] are considered to be local. Local symbols are local on a per-file basis. Each `INCLUDE` file (as well as the main file) can have its own locals that will not interfere with similarly named symbols of the remaining participating files. This has two advantages: First, symbols can be re-used in another `INCLUDE` file in a completely different way. Second, local symbols are not visible outside the file that contains them. This last benefit makes it possible to write quite complex `INCLUDE` files while making only the global variables and subroutine entry labels visible to the outside.

Marking big blocks as comments

#IFDEF without any expression following will always evaluate to False. This can be used to mark out a large portion of defunct code or comments. Simply «wrap» those lines within **#IFDEF** and **#ENDIF** directives. This saves you the trouble to individually mark each line as comment, eg.,:

```
#IFDEF  
  
This is a block of comments explaining all the little  
details of this great assembly language program..  
Blah, blah, blah..  
  
#ENDIF
```

The only drawback is that the listing file will not include this section. In some cases this is desirable, in others it isn't.

Creating 'menus' of possible -D option values

ASM8 -Dxxx [[-Dxxx]...] is used to pass up to ten symbols to the program for conditional assembly. Here's a tip for creating 'menus' of possible symbols to use with the **-D** option, so you don't have to remember them. An example follows:

```
#ifdef ?  
  #message *****  
  #message * Choice of run-time conditional symbols  
  #message *****  
  #message * DEBUG: Turns on debugging code  
  #message * JL: Target is MC68HC908JL3  
  #message * GP: Target is MC68HC908GP32  
  #message *****  
  #fatal Run ASM8 -Dx (where x is any of the above)  
#endif
```

The command **ASM8 -D? PROGRAM.ASM** will display the above 'menu' of possible **-D** values and terminate assembly. If you make it a habit of doing this in all your programs, then at any time you're not sure which conditional(s) to use, simply try assembling with the **-D?** option and you will get help. (A question mark is the smallest possible local symbol you can define. It is a perfect candidate for this job as it is easy to remember because it's like asking for help, and also because it is only visible in the main file. You could, of course, use any other symbol name you like.)

Using -Dx with specific values

You may also assign a specific value to a symbol defined at the command-line. This makes it possible, among other things, to assemble a program at different locations on the fly. For example, the following program:

```
;SAMPLE.ASM
#ifndef ROM ;needed to avoid «Duplicate symbol..» errors
ROM      equ      $F800      ;Default ROM location
#endif

Start    ORG      ROM
         rsp                      ;the program begins here
         ...                    ;rest of program is left to imagination
         bra      *              ;the program ends here
```

will be assembled at \$F800 with the command `ASM8 SAMPLE` but you could also assemble with a command similar to this: `ASM8 SAMPLE -DROM:$8000` to move ROM to a different location at assembly time.

As another example, you could declare an array where you define the dimension during assembly. No need to edit the source.

```
;SAMPLE.ASM
#ifndef ARRAYSIZE ;needed to avoid «Duplicate symbol..» errors
ARRAYSIZE equ    10          ;Default size for array
#endif
#if ARRAYSIZE < 2 ;check for minimum size allowed
    #error ARRAYSIZE must be at least 2
#endif

Status   ORG      RAM
         RMB      ARRAYSIZE
Pointer  RMB      ARRAYSIZE*2
         ...

Start    ORG      ROM
         rsp                      ;the program begins here
         ...                    ;rest of program is left to imagination
         bra      *              ;the program ends here
```

Getting rid of the «No ORG...» warning

If you are annoyed by the «No ORG...» warning that shows up whenever the assembler attempts to produce code or data without first having encountered an `ORG` statement, here's how to turn it off without actually specifying a fixed origin.

Somewhere before any code or data, and regardless of the current segment, use the pseudo-instruction:

```
org *
```

This is a «No Operation» `ORG` statement because it will simply use the current location counter for the `ORG`. It effectively does nothing. It will, however, set the appropriate internal flag that tells the assembler an `ORG` has been used and, thus, no warning!

Calling ASM8 from PE's WinIDE

Here's how to setup the WinIDE to use `ASM8.EXE` instead of PE's assembler. This will let the IDE catch possible errors during assembly and take you to the problem source file and line.

Create a batch file, called `ASM8.BAT`, which contains the following lines:

```
@echo off
c:\utils\asm8.exe %1 %2 %3 %4 %5 %6 %7 %8 %9 -T+ >c:\temp\error.out
```

where `c:\utils` is the directory `ASM8.EXE` is located and `c:\temp` is a directory you want to use for sending the temporary file with errors.

In WinIDE, select "Environment -> Setup Environment -> Assembler/Compiler -> EXE Path"

Put the path of the `ASM8.BAT` file you created earlier.

In the box Type, select "Other Assembler/Compiler"

In Options, type: `%FILE%`

Select the checkboxes for "Wait for compiler to finish", "Save files before assembling", and "Recover Error from compiler" and deselect the remaining ones.

The Error Filename should be `c:\temp\error.out` (or whatever filename you actually used in your `ASM8.BAT` above)

The Error Format should be "Borland Compatible"

Tips on using the MEMORY directive

The `MEMORY` directive is generally very useful for any program. It could help you save precious debugging time by alerting you whenever you accidentally put code and/or data where there is no real or available memory. The best place to use this directive is the same include file that defines the particulars of a specific MCU. And, assuming you always `INCLUDE` one such file in every program you write, you can forget about it.

Another use for the memory directive is to help you write a program that does not necessarily reside in specific memory locations but, rather, it occupies no more than so many bytes. For example, you're writing a small program that must be no more than 100 bytes long. Here's how to set the `MEMORY` directive to warn you should you go over this limit:

```
Start      rsp                ;the program begins here
           ...                ;rest of program is left to imagination
           bra      *          ;the program ends here

#memory Start Start+100-1    ;allowed range = Start to 100 bytes later
```

If while writing your program you begin getting MEMORY Violation warnings, you'll know you have reached (actually, gone beyond) the allowed limit. You must cut down the size of your code until the warning disappears.

Linux/Win32 version addendum

The DOS/Win and Linux versions are practically identical. This document covers the DOS/Win version.

A few differences with the Linux or Win32 versions are listed below:

For the Linux version, you must keep the included `asm8.cfg` in the same directory as the `asm8` binary (for example, `~/bin` for Linux or `C:\Utils` for Win32, etc.). Any time you change options and save them (with the `-W` option) a new `asm8.cfg` file will be created in the current directory (or updated if it already exists). If you want to make this new project the default, leave in the project's directory, and run `asm8` from there.

In case you also want to make this the new global default, "mv" (Linux) or "move" (Win32) it to the `~/bin` or other directory where your `asm8` binary is, and anytime a local `asm8.cfg` isn't found, the global one will be used instead.

Another difference from the DOS/Win version is that filenames are case-sensitive. But, to ease porting from DOS/Win, if a file (e.g., `INCLUDE`) is not found, it will be searched for again as "all lowercase" and, if not found a second time, it will be searched for once more as "all uppercase." This makes it easier to transfer files from DOS/Win to Linux and not have to rename them, or do so but not have to also change the source code.

These are pretty much the only differences in behavior. Program syntax is identical for all platform versions.