1 qFORTH Compiler

By using the MARC4 qFORTH compiler, embedded-system designers no longer have to stick to assembly language; the compiler generates a highly optimized object code. The smart qFORTH compiler translates your high-level qFORTH program into the MARC4 processors native code. The compiler system selects the right assembly-language instruction sequences and addressing modes for optimal operation. The intermediate code passes through rule-based expert systems at different optimization stages. This code is optimized for local centers of reference (colon definitions, macros, loops) to minimize stack operations and register references; it actually scoreboards register references to eliminate redundancies.

The qFORTH compiler supports standard FORTH constructs such as: BEGIN .. AGAIN, BEGIN .. UNTIL, CASE .. ENDCASE, DO .. LOOP, IF .. THEN, IF .. ELSE .. THEN, BEGIN .. WHILE .. REPEAT, and the following 4-bit and 8-bit data types: constants, variables, arrays and 8-bit ROM look-up tables. qFORTH extensions are interrupt functions, direct I/O port access, in-line assembly language and direct register access. The compiler also generates a line-number reference file to support source-code debugging in the MARC4 simulator and emulator.

The compiler is available in two versions. The fully integrated version is run by selecting 'Compile' in the menu bar of the MARC4 integrated environment menu and the command-line version is run by typing QFORTH2, followed by options and the name of the file to be compiled at the DOS command line.

1.1 The qFORTH Program Structure

In order to compile your qFORTH program correctly, the compiler expects the program to be composed of directives, definitions and statements. Most qFORTH programs will contain at least a group of statements which will perform computational operations. These statments are edited according to the guidelines outlined in the qFORTH Programmer's Guide. Whether or not you add compiler directives and CONSTANT definitions is dependant on the requirements of your program. They are more or less optional when compiling a qFORTH program. Parameters are expected by the compiler, but not defined by the programmer. The compiler will substitute default values such as for stack size allocation.

At the end of this chapter you will find a section which lists the default values used by the qFORTH compiler. But first it is necessary to re-examine what the three sections are which make up a qFORTH program.

1.1.1 Compiler Directives

The directives are compiler switches used to control the way in which your program is compiled and to specify the format of your compiler generated file(s). The majority of the directives can be implemented as in-line commands appearing at the beginning of your program code.

1.1.2 Definitions

The CONSTANT and VARIABLE definitions which are values referenced by your program via names. They should be assigned before the CONSTANT or VARIABLE is referenced within the program.

1.1.3 Statements

A qFORTH program is composed of various statements grouped together to perform a particular task which your program invokes via a word. These words are called **colon definitions** because they appear in your qFORTH program as starting with a colon ('**:**'), followed by a space and the name assigned to these group of statements. A statement group is a sequential list of MARC4 instructions, words found in the qFORTH system library or words which have been defined in your program before invoking this subroutine.

Note: All colon-definitions end with a semi-colon ('**;**').

Sequences of functionally grouped words are called modules. Modules used to perform the underlying computational tasks of the MARC4 are often caused by interrupt service routines. These are predefined names according to the naming conventions described in the qFORTH Programmer's Guide and are identified by '**: INT<x>**', whereby <x> is replaced by the priority number 0 to 7.

The program entry point is identified as the **$RESET** service routine since it is the first word which the MARC4 processor will execute after power-on reset. Normally, this colon definition is located at the end of your source program and consists of two parts: the register and the application initialization section. After the initialization of the stack pointers, the on-chip peripherals and the RAM variables of the application have to be put in a well-defined state.

1.1.4 Kicking the Assembler Habit

This short description has been intended as an overview to program composition as required by the qFORTH compiler.

To achieve a tighter code with your high-level language, remember the following rules and apply them more or less in order.

- Rethink your approach to problems to see if you can't find a more elegant solution.
- Make sure you are storing and manipulating your data efficiently. Accessing data using a pointer requires almost three times the number of instructions required to access the same information using array indexing.
- Make your code less abstract and take advantage of hardware-specific shortcuts wherever possible, always weighing the tradeoffs between speed and development time.
- Optimize your algorithms, eliminating all redundant and unnecessary operations. Use the address activity profiler in the emulator or simulator and optimize where it will do the most good.
- To maximize the limited on-chip RAM, minimize the usage of local variables and too much nested subroutine calls.
- To reduce the stack usage, check your parameter passing and subroutine nesting as well as the number of concurrent interrupt service routines.
- Use assembler instructions for the time-critical code but do not fall back on writing whole modules in assembler.

Stick to these approaches and you will be writing applications that will keep your competition awake at night, not you.

1.2 Using the Compiler

Check that the correct directory path for qFORTH has been entered in the setting window 'Directories' (see installation guide).

- Edit your program file(s)
- Setup the project's file name
- Setup the compiler options
- Invoke the compiler

To set the project's filename use the pull-down menu 'Compile' and select 'Set project file'. The project's filename means the leading filename of the project which will be compiled (see figure 1).

To invoke the compiler, use the pull-down menu 'Compile' and select 'Built Project' or press the key **<F9>**. This occurence will compile the whole project.

The 'Compile' pull-down menu is shown in figure 2. If you wish to compile the currently edited file then either press **<Alt-C>** followed by the carriage return key or simply enter **<Alt-F9>** from within the editor. This will automatically start the compiler using the active file as its input filename.

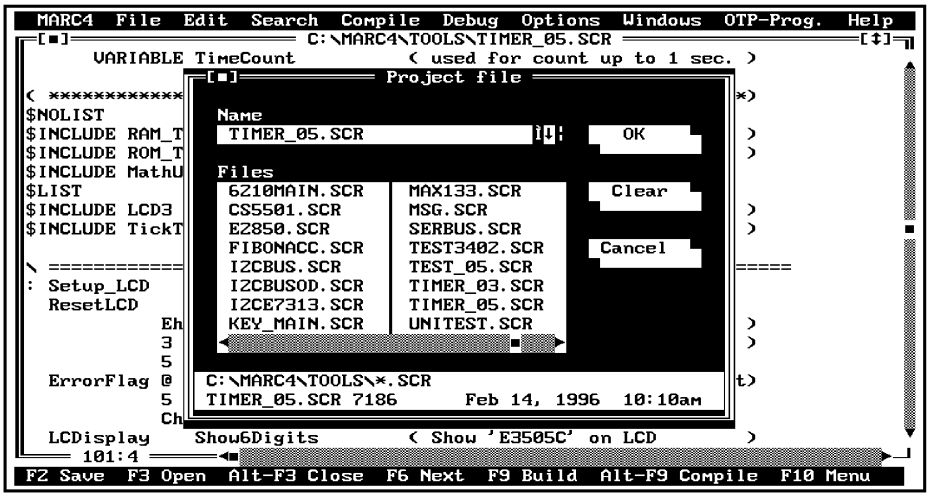
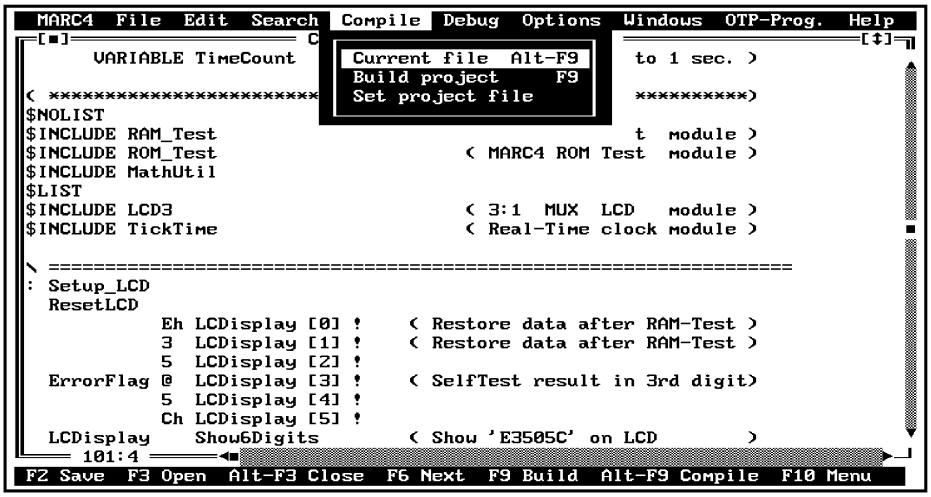


Figure 1. Set-up of project file to be selected first



12519

Figure 2. Selection of the compiler within the environment

1.2.1 Compiler Generated Messages

If the compiler detects a code which can not compile correctly, a warning or an error message will be displayed. The occurence of a warning is an indication to you that your program will still compile and is executable, however, it may not produce a code with the desired kind of execution. If an error is found, the compiler will terminate since it is unable to generate executable code. A complete list of all warning and error messages can be found in the Appendix.

The information given during any compilation is the following:

- The qFORTH compiler version and the date of creation with the qFORTH system library used with their date of creation
- The name, drive and directory path of the compiled source file
- The optimizer passes, because a '.' is written to the screen for each step during optimization and a ',' when macro expansion takes place.
- The compilation result:

  If no errors were found, the amount of ROM (in bytes) required and the calculated CRC check-sum stored in the last two bytes of the ROM is displayed.

  If errors occur during the compilation, the error and/or warning messages will be reported instead. They will be attached at the end of your source code within the list file.

Note: A complete list of all warnings and error messages can be found in chapter 4.6 "Error and Warning Messages".

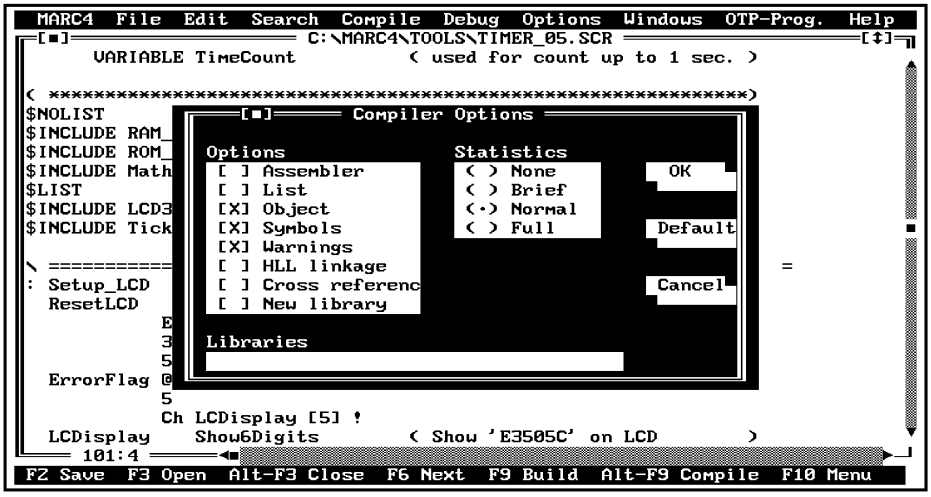
1.2.2 Compiler Generated Files

The compiler generates various files which are normally directed to the same filename, drive and directory path as the project's source file (see table 1).

Table 1 List of all compiler generated files

Extension	File Type & Contents	Format
HEX	Object code	Binary
SYM	Symbol table	Internal
LST	Complete list and statistics	Text
CRF	Cross reference file	Text
ASS	Assembly code list file of compiler generated object code	Text
HLL	Line number reference file for high level language orientated debugging	Internal
LIB	User generated library for often used routines	Internal
RPT	Compilation success/error report file within SDS	Internal

1.2.3 Compiler Switches



12520

Figure 3. Default setting for compiler switches within the option menu

To set the compiler switches for different compiler options, use the pull-down menu "Options" and select "Compiler" within the SDS2 environment.

Compiler Options

Assembler

This controls, whether an assembler list file is generated. The default extension is "**ASS**", the default filename is that of the source file. This output file may be used to check the efficiency of the generated object code.

List

This controls whether a source listing has to be generated. The default extension is "**LST**", the default filename and path is that of the source file. This generated file contains all events during compilation, depending on additional compiler switches.

Object Default setting

This controls whether a binary object code file has to be generated. The default ectension is "**HEX**" and default filename is that of the source file. By default, an object and symbol file with full optimization is created.

Symbols Default setting

This controls whether a symbol file has to be generated. The default extension is "**SYM**" and the default filename is that of the source code. This file is necessary if you want to check your code with all defined symbols (subroutines and variables). By pressing the function key **<F7>** at the software simulator or emulator, you can view the symbol table data.

Warnings Default setting

This controls whether warnings are written onto the screen and with the setting of the additional switch "**List**" into the list file too.

HLL linkage

This controls whether a high-level-language debugger link file has to be generated. The default extension is "**HLL**", the default filename and path is that of the source file. This generated file enables source level debugging (see chapter 5 "Software Simulator").

Cross reference

This controls, whether a cross reference file has to be generated. The default extension is **"CRF**", the default filename and path is that of the source file. The cross reference file shows the correlations of all used symbols (subroutines, variables and constants) with regard to their definition and their use for different source files.

New Library

This controls whether a new user library has to be generated. The default extension is **"LIB"**, the default filename is that of the first source file. When a user library is generated, no object, symbol and assemblerfile will be created. A user library contains all codes generated during this compilation or all codes read from other user libraries (see input line "LIBRARIES").

Compiler Statistics

None

By setting this switch, all statistical information is suppressed in the list file.

Brief

Generates a summary of all errors, all defined words and all defined variables at the end of the list file.

Normal Default setting

Additionally lists the return and expression stack usage of all routines, the addresses of all words placed in ROM, a summary of left ROM holes, unused RAM nibbles and unused short call address entries, an overview of bytes saved during the optimization steps and information about the compiler's memory usage.

Full

Additional information about the subroutine placement algorithm, the CPU time for the different compilation steps, statistics on the usage of the internal symbol table data base, summary of created files and used compiler switch settings.

Libraries

This input line controls whether one or more user libraries have to be read after the system library has been read. By default, no additional user library is read. The list may consist of up to 7 user libraries, their names must be separated by a comma. The default extension is "**LIB**".

1.3 Compiler Directives

A compiler directive may occur anywhere in the source file(s), the first character of a compiler directive is always an "$". In general, a directive is used to control the compilers behavior when processing the source code. Compiler directives can not be abbreviated.

1.3.1 Conditional Compilation

To make your job easier, qForth offers conditional compilation. This means that you can decide what portions of your program to compile based on defined symbols.

The conditional directives are similar in format to the compiler directives you are accustomed to. In other words, they have the format.

$directive <arg>

Where **directive** is the directive (such as **DEFINE**, **IFDEF**, and so on), and **<arg>** is the argument, if any.

Note: There must be a blank as seperator between directive and **<arg>** .

List of conditional compilation directives:

$DEFINE <symbol>	Defines symbol for other directives

To define a symbol, insert this directive into your program. **<symbol>** follows the usual rules for identifiers as far as length, characters allowed, and other specifications are concerned.

Example: **$DEFINE Debug**

This defines the symbol 'Debug' used for the remainder of your program which is to be compiled.

$IFDEF <symbol>	Compiles the following code if **<symbol>** is defined
$ELSE	Compiles the following code if the previous **$IFDEF** is not true, i.e., the **<symbol>** is not defined.
$ENDIF	Marks the end of **$IFDEF** and/or **$ELSE** section.

```
Example:    $IFDEF      <symbol>
                        <source code A>
            $ELSE
                        <source code B>
            $ENDIF
```

Where **$IFDEF** is followed by the appropriate argument, and **<source code>** is any amount of qFORTH statements. If the **<symbol>** is not defined, the **<source code A>** is ignored as if it had been commented out of your program.

Within a skipped conditional block only **$IFDEF**, **$ELSE** and **$ENDIF** are processed. All other words (including directives) are ignored. Skipped conditional blocks are marked with a hash sign '#' in the listing file.

Often you have alternate chunks of source code. If the symbol is defined, you need to compile one chunk, and if it's false, you need to compile the other chunk. The qFORTH compiler enables you to do this with the **$ELSE** directive.

Note: All $IFDEF directives must be completed within the same source file, which means they cannot start in one source file and end in another. However, an $IFDEF directive can encompass an include file.

```
Example:    $IFDEF MUX4–LCD
            $INCLUDE LCD–MUX4.SCR
            $ELSE                   \ otherwise 3:1 MUX
            $INCLUDE LCD–MUX3.SCR
            $ENDIF
```

In this way, you can select alternate include files based on the same condition. You can nest **$IFDEF .. $ENDIF** constructs to achieve the following results:

```
$IFDEF Version–2
            <source code A>
$IFDEF Debug
            <additional code>
$ENDIF      \ end of debugging output
            <source code B>
$ENDIF      \ Version–2
```

1.3.2 Compilation Control

Index Checking

$I+ Default setting
$I–

Normally the index checking for the array indices is on. When the default setting **$I+** is active during compilation, the constant array indices must be kept in the range 0 to <length-1>.

By using **$I–** for a section of code, the index checking is switched off, i.e., any constant array index may be specified. For example, specifying the **DataArray [-1]** could be useful while writing to the array using **[+Y]!** or **[+X]!** instructions within a loop.

Macro Expansion Control

$EXPAND Default setting
$NOEXPAND

To modify the time of the macro expansion and thereby the amount of optimization done by the compiler, the $EXPAND and $NOEXPAND directives may be used. The use of the directives $EXPAND or $NOEXPAND on the outside of a CODE definition sets this directive globally. This means that the macro expansion mode influences all following CODE definitions.

By default all macros are expanded before the optimization process is started. The directive $NOEXPAND means that CODE definitions are expanded after the optimization process has finished.

Branch Stripping Algorithm

$BRA_STRIP NOTALL Default setting

Unconditional branches are stripped so that short branches will stay short branches, i.e., if a short branch leads to a second unconditional short or long branch, the first short branch could be stripped. If this results in a long branch, stripping is suppressed.

$BRA_STRIP ALL

All branches are stripped, regardless of whether short branches could become long branches. This kind of branch stripping may result in an increase in code length, but will minimize the execution speed.

ROM CRC-Algorithm

$CRC <arg>

The **$CRC** directive (**C**yclic **R**edundance **C**heck) checks the contents of ROM. The check sum will be stored after compilation at the last two ROM bytes of the last physical ROM bank.

The following arguments are available:

DEFAULT	16-bit software CRC
SIMPLE	8-bit software CRC (optimized code size)
HARDWARE	16-bit hardware CRC (for MARC4 variants with built-in selftest)

1.3.3 List-File Directives

The list file directives will only have an effect, if **/LIST** was specified in the command line or as one of the compiler options in the integrated environment.

$NOLIST Default setting
$LIST

The source listing is suspended by **$NOLIST** until **$LIST** is found again in the source code.

$PAGE

$PAGE will force a form feed in the print output file, if the list output is active.

$DEBUG_STACKS

The compiler directive **$DEBUG_STACKS,** when included in one of the source files, writes the calculated expression and return stack effects of all code and colon definitions into the print file. The four columns following the source line number contain stack depth values that are relative to the beginning of this source line.

The sequence of the columns is as follows :

- current number of nibbles on the expression stack,
- current number of used return stack entries,
- maximum expression stack depth reached within this routine (nibbles),
- maximum return stack depth reached within this routine.

```
27      $DEBUG_STACKS
28                                  : INT7
29 |   5 |   1 |   5 |   1 |              PortData @
30 |   6 |   1 |   7 |   1 |              Port0 OUT
31 |   5 |   1 |   7 |   1 |  ;
32
33
34                                  $NOEXPAND
35                                  $OPTIMIZE  –XYTRACE
36                                  CODE X–
37 |   0 |   0 |   0 |   0 |              [X–]@  DROP
38 |   0 |   0 |   1 |   0 |  END–CODE
39                                  $OPTIMIZE  +XYTRACE
40
41
42 |   0 |   0 |   0 |   0 |              >SP  S0
43 |   0 |   0 |   0 |   0 |              >RP  NoRAM
44 |   0 |   0 |   0 |   0 |
45 |   0 |   0 |   0 |   0 |  Port0  IN  0 =
46 |   0 |   0 |   2 |   0 |  IF         RAM_TEST
47 |   0 |   0 |   0 |   0 |              ROM_TEST
48 |   0 |   0 |   7 |   3 |  THEN
49 |   0 |   0 |   7 |   3 |
50 |   0 |   0 |   7 |   3 |  0  0  Timer_A  2 !
51 |   0 |   0 |   7 |   3 |
52 |   0 |   0 |   7 |   3 |  PortData  X!  X–
53 |   0 |   0 |   7 |   3 |  8 #DO
54 |   0 |   1 |   0 |   1 |              0 [+X] !
55 |   0 |   1 |   7 |   1 |  #LOOP
56 |   0 |   0 |   7 |   3 |
57 |   0 |   0 |   7 |   3 |  2_Hz  Prescaler  OUT
58 |   0 |   0 |   7 |   3 |  ;
```

All values related to the return stack are counted as 16-bit or 4 nibbles entries. The MARC4 core uses 12-bit words on each return stack entry. The address space of the fourth nibble, not used by the return stack, will be assigned by the compiler for single 4-bit variables.

All calculated values are relative to the start of the CODE or colon definition. The expression stack values always start with 0. The return stack value starts with 0 in CODE definitions. In colon definitions it starts with 1 because of the return address which is already saved on the return stack.

1.3.4 Stack Effect Directives

The following directives have no effect if the compiler switch **WARNINGS** is turned **OFF**. The warning messages of the compiler are very helpful when looking for an unexpected expression stack under-/overflows or i.e., different stack effects of **IF .. ELSE .. THEN** parts.

On the other hand, the programmer may be aware of the fact that i.e. a LOOP block eats up a specified number of elements from the stack. Therefore, if the programmer is sure that this particular code works perfectly, the compiler warnings can be turned **OFF**. These compiler directives will be placed at the end of each 'block' of qFORTH words (i.e., a **DO .. LOOP**).

They always start with '**[** ' and end with the symbol ' **]**'. In between those two symbols each combination of the following directives are allowed:

E <number>	Define expected expression stack effect,
R <number>	Define expected return stack effect,
Ex <number>	Define maximum expression stack effect,
Rx <number>	Define maximum return stack effect,
?	Return and expression stack effects of the previous block are unknown, the corresponding **WARNING** message will be turned **OFF**.

Example:

```
$I-                         \ Turn index checking OFF
: BCD@                      \ Push a number of
                            \ digits onto the stack
    Multiplier [ –1 ] Y!    \ Setup array pointer
    8 #DO
        [ +Y ]@             \ Push an array
                            \ element onto the stack
        [ E 0 ]             \ Turn the compiler
                            \ warning message OFF
    #LOOP
    [ E 8 ]                 \ Set correct number
                            \ pushed onto stack
;
$I+
```

1.3.5 Optimization Control

The amount of optimization done during the compilation process can be controlled by the **$OPTIMIZE** control switch. By default all optimization steps will be performed.

$OPTIMIZE <switch1>, <switch2>

The ABSOLUTE range of optimizations to be performed is set by qualifying the control switch **$OPTIMIZE**. The only types of optimization performed furthermore are those, that are listed after **$OPTIMIZE**.

$OPTIMIZE {+ |–}<switch1>, {+ |–}<switch2>

The optimization qualifiers can also be used in conjunction with the **$OPTIMIZE** control switch for a

RELATIVE setting in the source files. The kind of optimizations performed is determined by adding (+) or removing (–) the listed types from the current optimization set.

$OPTIMIZE ?

The current optimization control settings are written into the print output file.

$NOOPTIMIZE Default setting
$OPTIMIZE

All kinds of optimization are inhibited when the **$NOOPTIMIZE** is specified. Whereas **$OPTIMIZE** will cancel a previous **$NOOPTIMIZE** directive, i.e., the optimization set is the same as before the **$NOOPTIMIZE** directive.

It is possible to control the optimization process in such a way that some specific subroutines or macros will not be optimized. For example, no register tracing in a hand-optimized memory block MOVE routine.

$OPTIMIZE Qualifieres

The user may parameterize the **$OPTIMIZE** directive with the following qualifieres:

CALL	CALL optimizer pass [CALL →SCALL],
BRANCH	Branch optimizer pass [BRA → SBRA],
CMP	Comparison optimizer,
DROP	DUP .. DROP optimizer,
SWAP	SWAP .. SWAP optimizer,
XYLOAD	Register load optimizer,
XY@!	Register load with memory fetch/store operation,
XYTRACE	Register scoreboarding, preincrement /postdecrement

XYLOAD

Sequences like **LIT_p LIT_q .. X!** will be optimized to a **>X $pq** instruction.

XY@!

Sequences like **>X $pq .. [X]!** will be optimized to a **[>X]! $pq** instruction.

XYTRACE

By reloading the **X** or **Y** register sequences like **[>X]@** or **[>Y]! $pq** will be replaced by **[+X]@** or **[Y-]!** operations, whenever possible.

CMP

Sequences like **CMP_cc** .. **TOG_BF .. BRA** are optimized to the sequence **CMP_**$\overline{cc}$ .. **BRA**, where $\overline{cc}$ is the opposite condition of **cc**. Also **TOG_BF .. TOG_BF** sequences are omitted which may result from macro expansions.

CALL

A **CALL** instruction is replaced by a **SCALL**, whenever possible.

SAVECONTXT

The **INTx** prefix and postfix register save macro (**X@ Y@ CCR@ .. CCR! Y! X!**) is reduced, whenever possible. If **INT5** does not change the register, **X@** and **X!** are removed from the routine's prefix and postfix sequence.

The lowest priority interrupt routine may be compiled with:

```
$OPTIMIZE –SAVECONTXT
: INT0      Calculate_On_Off
            Update_LCD
;
$OPTIMIZE +SAVECONTXT
```

BRANCH

A long branch instruction is optimized to a short branch instruction within a code page whenever possible.

BRA_EXIT

Unconditional branches to an **EXIT** instruction are replaced by an **EXIT**, also unconditional branches to an instruction that is placed directly before an **EXIT** are replaced by this instruction followed by an **EXIT**.

BRA_STRIP

A branch to a second unconditional branch will be changed so that the first branch goes directly to the target of the second branch. This will not save any code, but result in a faster execution speed. See also the compiler directive **$BRA_STRIP**, which allows you to control the amount of branch stripping being performed.

DROP

Any sequence **<Push nibble onto stack>** .. **DROP** will be removed from the code if this nibble is not used anywhere else and results in no side effects.

Note: Because **[+Y]@ DROP** will change the Y register, it is not optimizable.

SWAP

Any sequence **SWAP .. SWAP** will be removed whenever possible. Furthermore, any sequence **LIT_x .. LIT_y .. SWAP** will be optimized to **LIT_y .. LIT_x**.

1.4 Compiler Optimization Steps

The previous section described how to use the compilers optimization directives. The code optimizations implemented are reviewed in this section.

1.4.1 Branch Optimizer

Short branches are used whenever the address is achieveable within the present 64-byte page, otherwise full branches are used. The programmer does not need to be aware of any page boundaries.

1.4.2 Call Optimizer

Short calls can only be used for colon definitions in the Zero Page (the first 512 bytes). These definitions are automatically selected to be placed in the Zero Page as a result of their size and static usage. The programmer can force a Zero Page placement by appending either **AT <Address>** or **'[Z]'** compiler directives at the end of a colon definition.

1.4.3 Peephole Optimizer

The peephole optimizer replaces a sequence of instructions with a shorter, more efficient sequence. In general, a stack architecture allows a much wider peephole than normal, as stack effects within a 'basic block' may be evaluated at compile time. This means that a given code sequence does not need to be consecutive. Currently eight separate peephole sequences are checked. The following example shows the two sequences which were found to occur most frequently.

Example 1: Compile time constant folding

Source	Assembly code	Optimized code
FRED @	Lit_3 Lit_4 X! [X]@	[>X]@ $FRED

Example 2: DUP DROP optimizing resulting from the MARC4 implementation of the compare instructions, where only one of the top two elements is dropped.

Source	Assembly code	Optimized code
DUP 3 =	DUP	Lit_3
IF	Lit_3	CMP_NE
..	CMP_EQ	SBRA $THEN
THEN	DROP TOG_BF BRA $THEN	

1.4.4 Register Tracking

While a good assembly code programmer may never write code with redundant **DUP** and **DROP** instructions, it is often the case that he may forget exactly which variables and addresses are cached in registers. A good compiler however, can keep track of which register contains are variable. This is especially true in qFORTH since the programmer's model of the machine has no additional registers.

Example 1: Variables **FRED** and **BERT** are in consecutive RAM locations

Source	Assembly code	Optimized	Final code
FRED@	Lit_3	[>X]@ $FRED	[>X]@ $FRED
BERT +!	Lit_4	[>Y]@ $BERT	[+X]@
	X!	ADD	ADD
	[X]@	[Y]!	[X]!
	Lit_3		
	Lit_5		
	Y!	(+! macro)	
	[Y]@		
	ADD		
	[Y]!		

Sometimes register tracking may also eliminate redundant address register loads across an IF statement.

Example 2:

```
FRED @ DUP 5 <>
IF              BERT !
ELSE            DROP
                0 FRED !
THEN
```

1.5 The Command-Line Compiler

Compiling qFORTH programs can also be done by using the command-line approach common to most computers where each step in program generation occurs from the command line. On your PC this means from the DOS command line indicated by the prompt, such as the drive indicator.

C:\MARC4 > qFORTH2 [/<switch>]
<filename>[/<switch>]

To invoke the compiler, enter the program name **qFORTH2** followed by the filename to be compiled. Normally, a file extension is not required since **'SCR'** is default when compiling a main program.

As an example, to compile a file called 'MYFILE.SCR' with the generation of a list and object code file, the following command-line sequences would be accepted as valid by the compiler:

QFORTH2/LIST/NOSTAT MYFILE
QFORTH2 MYFILE/LIST/STAT=NO
QFORTH2/LIST/SYM MYFILE/STAT=FULL

An overview of the various compiler switches, options and directives accepted by the command-line compiler is listed in the subsequent sections of this chapter.

After the compilation of your program is completed, the DOS drive indicator will appear on the screen permitting you to either enter the simulator or emulator (in command-line mode) or to go back to your program editor to correct any possible errors which may have occured.

1.5.1 Compiler Generated Messages

The generated messages of the command-line compiler version are the same as the MARC4 environment integrated version.

1.5.2 Compiler Generated Files

A listing of all generated files is shown in table 1.

1.5.3 Setting the Compiler Switches

The compiler switches of the command-line version are the same as the integrated version. Default switch settings do not have to be called in the command line. For more detailed information, see the section 'Compiler Switches' of the integrated version.

Object Code Generation

/NOOBJECT

/OBJECT[=<object file>] **Default setting**

This controls whether a binary object code file has to be generated. The default extension is **".HEX"** and the default filename is that of the source file.

/NOSYMBOLS

/SYMBOLS[=<symbol file>] **Default setting**

This switch controls whether a symbol file has to be generated. The default extension is **".SYM"** and the default filename is that of the source code. This file is necessary if you want to check your code with all defined symbols (subroutines and variables). By pressing the function key **<F7>** in the software simulator or emulator, you can view the symbol table data.

List File Generation

/LIST[=<list file>]

/NOLIST **Default setting**

This switch controls whether a source listing has to be generated. The default extension is **".LST"** and the default filename and path are that of the source file. This generated file contains all events during compilation, depending on additional compiler switches.

/NOWARNING

/WARNING **Default setting**

This controls whether warnings will be written onto the screen and – with the setting of the additional switch "$List" – in the list file, too.

/NOSTATISTICS

/STATISTICS[=<statistics qualifier>]

<**statistics qualifier**>:

/STATISTICS=NO (is identical to NOSTATISTICS)

/STATISTICS=BRIEF

/STATISTICS=NORMAL **Default setting**

/STATISTICS=FULL

No

By setting this switch, all statistical information is suppressed in the list file.

Brief

Generates a summary of all errors, all defined words and all defined variables at the end of the list file.

Normal

Lists additionally the return and expression stack usage of all routines, the addresses of all words placed in ROM, a summery of left ROM holes, unused RAM nibbles and unused short-call address entries, an overview of bytes saved during the optimazition steps and information about the compiler's memory usage.

Full

Additional information about the subroutine placement algorithm, the CPU time for the different compilation steps, statistics on the usage of the internal symbol table data base, summary of created files and used compiler switch settings.

Debugging Support File Generation

/ASSEMBLER[=<assembler file>]

/NOASSEMBLER **Default setting**

This controls whether an assembler list file will be generated. The default extension is **".ASS"**, the default filename is that of the source file. This output file may be used to check the efficiency of the generated object code.

/CRF[=<crossreference file>]

/NOCRF **Default setting**

This controls whether a cross reference file has been generated. The default extension is **".CRF"**, the default filename and path is that of the source file. The cross reference file shows the correlations of all used symbols (subroutines, variables and constants) with regard to their definition and their use in the different source files.

/LOG[=<HLL file>]

/NOLOG **Default setting**

This switch controls whether a high-level language debugger link file has to be generated. The default extension is ".HLL", the default filename and path is that of the source file. This generated file enables source-level debugging (see chapter 5 "Software Simulator").

Library Management

/NEWLIB[=<library file>]

/LIBRARY[=<library file>[,<library file>]]

This command controls whether one or more user libraries have to be read after the system library has been read.

/SYSLIB

Controls whether a new system library has to be generated or not. The default filename is 'qFORTH2.LIB', generated from the input source file. The source files to be compiled into a system library must have a certain format, otherwise the compilation will fail.

Note: This compiler switch is reserved for Atmel Wireless & Microcontrollers' internal use only.

1.6 Error and Warning Messages

Notes:

All errors marked with (****) are severe errors which indicate that the compiler does not work properly. In this case, you should send your source code which caused the error, together with your system library and a brief description, to Atmel Wireless & Microcontrollers.

DOS errors, which are preceeded by the word DOS, are not explained in this manual. Refer to your DOS manual. Turbo (Pascal) runtime (RT) errors are caused by an incorrect compiler source code.

1.6.1 Coded Error and Warning Messages

001 File not found

When including a file with the $INCLUDE directive, this file was not found. All files to be included are expected in the same directory as the source file, as long as there is no directory path preceeding the filename.

005 Turbo RT: Object not initialized (****)

TURBO runtime error caused by an incorrect compiler code.

006 Turbo RT: Call to abstract method (****)

TURBO runtime error caused by an incorrect compiler code.

050 WARNING — Source line too long. Truncated after 120 characters

A source line is always processed up to 120 characters only. Additional characters are ignored.

051 WARNING — End of file reached while scanning comment

When scanning comment, the end of file was reached prior to the end of comment. The closing parenthesis ')' seems to be missing.

052 Too many nested INCLUDE's. INCLUDE will be ignored.

Includes may be nested only 4 levels deep. Additional nested include files are ignored. Nevertheless, including can be done sequentially without limitations.

053 Numeric value out of range

The numeric value read was either out of the machine's integer number range, or an array index was out of its range. Arrays always start with index 0. This message is also issued if you force an object via 'AT' to a location outside of the current RAM or ROM address range.

054 Internal stack overflow (****)

The compiler's internal number stack has overflowed.

055 Internal stack empty (****)

The compiler's internal number stack was empty when the compiler tried to get a number from the stack.

056 Number expected

The compiler expected a number or constant as next item within the source file. This message is often seen on constant or array definitions following a look-up table. Please rearrange the sequence of definition so that a variable or colon/macro definition follows a look-up table.

057 Assembler definitions expected when creating library

When compiling a system library source, the compiler always expects a section with assembler definitions at the beginning. This section was not found.

058 Additional characters ignored

There are characters after the end of a program in your source file. They will be ignored by the compiler.

059 Only CODE, ':' or CONSTANT definitions are permitted

In a system library source, only CODE, COLON and CONSTANT definitions may occur.

060 END–ASSEMBLER expected

The end of the assembler section has to be marked with this word. The compiler did not find it in the system library source code.

061 QFORTH–LIBRARY expected

The COLON and MACRO definition in a system library source have to be enclosed by the words QFORTH–LIBRARY ... END–LIBRARY. This error occurs if there are no COLON or MACRO definitions whatsoever.

062 Reserved word QFORTH–LIBRARY not found, will be added

When compiling a system library source, a COLON or MACRO defintion was found before the word QFORTH–LIBRARY.

063 Unable to handle. Skipped to next

When looking for the beginning of an object definition, an unusable object definition was found. The compiler skips to the beginning of the next object definition.

064 ':' added

Whenever an undefined name is found, and the compiler looks for the beginning of a new definition, this name is regarded to be the name of a COLON definition where the user forgot to write the colon.

065 WARNING — Undefined Word

An undefined word was found within a COLON or MACRO definition .

066 Undefined label or label referenced outside of definition

All labels used within a COLON or MACRO definition have to be defined in this definition, unless the labels are maked as 'special labels' which begin with the two characters '_$'. If this error occurs, one or more labels within a definition were not defined.

067 END–CODE expected

When compiling a macro, the beginning of the next definition was found while the macro was not compiled completely. In this case, an END–CODE is added by the compiler which causes the compilation of the macro to be completed properly.

068 ';' expected

When compiling a COLON definition, the beginning of the next definition was found while the colon definition was not compiled completely. In this case, a ';' is added by the compiler which causes the compilation of the colon definition to be completed properly.

069 WARNING — There is no special handling of negative numbers

The MARC4 is not able to process signed numbers in binary format. All negative numbers used in your source file will be treated as positive values.

070 $VERSION expected

The system library source code must begin with a $VERSION statement. The word $VERSION must be followed by a string that will identify this library version. The compiler also uses the version string to check the validity of user libraries.

071 Only 'CALL' and 'BRA' instructions permitted

When using assembler instructions in COLON or MACRO definitions, you are not allowed to use the short-call (SCALL) or short-branch (SBRA) instruction. The compiler will optimize the long-branch (BRA) and long-call (CALL) instruction to SBRA and SCALL instructions whenever possible.

072 Insufficient space for intermediate code (****)

When compiling a program, the code is stored in an intermediate array before the object code is assembled. This error does not occur if the space reserved within the compiler for intermediate code is defined to be large enough.

073 '%' or '$' not permitted in label names

These two characters may not occur in label names, for they are reserved to the compiler's use when substituting macros. Furthermore, care should be taken when using labels beginning with an underscore, for most labels in the qFORTH library begin with an underscore. This might cause duplicated label names.

074 WARNING — Label too long. Truncated to 16 characters

The length of a label is limited to 16 characters.

075 Duplicate label names

Within your program, two duplicate label names were found. You have to rename one of them.

Note: Avoid label names beginning with an underscore, as this might cause interferences with label names already used within the qFORTH library.

076 "]" expected

The option list or an array index must always be enclosed in square brackets. In an option list, these brackets must be preceeded and followed by at least one blank. When supplying an index, the opening bracket must be preceeded by at least one blank, the closing bracket must be followed by at least one blank. The index may be preceeded or followed by one or more blanks (optional). An index may only occur after an array name in the source code.

077 WARNING — Stack effect of word not computable

Normally, the compiler computes the EXP and return stack effects of every COLON and MACRO definition. This is impossible if

– BRA assembler instructions are used in the definition,

– you use a COLON or MACRO definition whose stack effects are un-computable,

– this COLON definition is recursive,

– this COLON or MACRO definition contains an IF–ELSE–THEN statement where THEN and ELSE part have different stack effects,

– this COLON or MACRO definition contains any loop (DO .. LOOP or #DO .. #LOOP or ... or BEGIN ... AGAIN or BEGIN ... UNTIL or ...) in whose block the RET or EXP stack effect is <> 0

– this COLON or MACRO definition contains a DO ... +LOOP statement wherein the RET stack effect is not 0 or the EXP stack effect is not 1.

–this COLON or MACRO definition contains a CASE statement wherein the effects of all selections are not the same. You can suppress this warning by classifying the COLON/MACRO as one, which stack effect do not has to be computed by supplying the option '?' or by explicitly typing the stack effects in the option brackets, e.g. [E 0 R 0].

078 Only ':' definitions can be forced to ZERO PAGE

Only COLON defintions can be forced to the zero page by the 'Z' option. A COLON definition forced to the zero page will be placed there, regardless whether it is called by any routine or not.

079 Nesting of object definitions not permitted

Object definitions may not be nested, i.e., you can not write a COLON or MACRO definition within an other COLON or MACRO definition.

080 ELSE or THEN expected; THEN will be added

When processing the THEN part of an IF statement, the beginning of another object definition or the end of the current definition was found. In this case, a THEN is added to correct the block structure.

081 THEN added

When processing the ELSE part of an IF statement, the beginning of another object definition or the end of the current definition was found. In this case, a THEN is added to correct the block structure.

082 #LOOP added

When processing an #DO ... #LOOP statement, the beginning of another object definition or the end of the current definition was found. In this case, a #LOOP is added to correct the block structure.

083 Last numeric entry omitted

When scanning the source code the compiler has to do a look-ahead of one word to process CONSTANT or ARRAY definitions. Therefore, when a number is found at the beginning of an object definition, the compiler has to read the next word to decide whether the number is valid or not. If a number is invalid, this is flagged at the word following the number with this message.

084 Predefined value is already initialized

Predefined constants like $RAMSIZE or $ROMSIZE can only be set once in a program's source code.

085 WARNING — Return stack doesn't start at address 0

The return stack does not start at adress 0, because it was forced to another location by 'AT'. This means, when an RET stack underflow occurs, NO SLEEP mode is entered and program exectution will continue at a random location. You should ensure that this mode is impossible when forcing the RET stack to a specific address.

086 $RAMSIZE value is insufficient

By declaring too large stacks or too much arrays or variables, there is insufficient space in the internal RAM to place all objects into it. The compiler has to be told the RAM size in the predefined constant $RAMSIZE, or a default value of now 111 nibbles is used.

087 AT not permitted here

The AT part of an array or a variable definition has to stand in front of the ALLOT part.

088 A label became too long when macroing

Calling macros in other macros over several levels may cause the label name length to overrun the limit. You should use shorter names or less excessive macro-in-macro-calls.

089 LOOP added

When processing a ?DO/DO ... LOOP statement, the beginning of another object definition or the end of the current definition was found. In this case, a LOOP is added to correct the block structure.

090 WARNING — THEN and ELSE block with different stack effects

When processing a COLON or MACRO definition, an IF–THEN–ELSE statement with different stack effects in the THEN and ELSE part was found. This causes the stack effects of the current COLON/MACRO definition to be uncomputable. An IF–THEN–ELSE statement with an absent ELSE part is regarded as an IF–THEN–ELSE statement with an RET and EXP stack effect of 0 in the ELSE part.

091 WARNING — RET stack effect in LOOP block is <> 0

The current COLON/MACRO definition contains any kind of loop in which the RET stack effect is not 0. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

092 WARNING — EXP stack effect in LOOP block is <> 0

The current COLON/MACRO definition contains any kind of loop in which the EXP stack effect is not 0. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

093 WARNING — EXP stack effect in final +LOOP block is <> 1

The current COLON/MACRO definition contains a DO ... +LOOP statement in which the EXP stack effect of the last block in front of the +LOOP is not 1. This causes the stack effects of the whole COLON/MACRO definition to be uncomputable.

094 UNTIL, WHILE or AGAIN expected. UNTIL added

When processing a BEGIN statement, the beginning of another object definition or the end of the current definition was found. In this case, an UNTIL is added to correct the block structure.

095 REPEAT added

When processing a BEGIN ... WHILE ... statement, the beginning of another object definition or the end of the current definition was found. In this case, an UNTIL is added to correct the block structure.

096 ENDCASE added

When processing a CASE .. OF .. ENDOF .. statement, the beginning of another object definition or the end of the current definition was found. In this case, an ENDCASE is added to correct the block structure.

097 ENDOF added

When processing a CASE .. OF .. statement, the beginning of another object definition or the end of the current definiton was found. In this case, an ENDCASE is added to correct the block structure.

098 System library incomplete! Contact Atmel Wireless & Microcontrollers for immediate support (****)

One basic part of the system libray QFORTH.LIB was not found. This error only occurs if your system library has been damaged by hard-disk errors. For first aid, delete qFORTH2.LIB and dearchive this file from MARC4.ARJ on the installation disk.

099 Internal compiler error ! Contact Atmel Wireless & Microcontrollers for immediate support (****)

Supply your source code for failure analysis.

100 $AUTOSLEEP and $RESET have fixed ROM addresses, AT ignored

The $AUTOSLEEP routine is always placed at ROM address 000h, while $RESET is placed at ROM address 008h. Trying to force these routines to different start addresses will result in this warning.

101 Symbol longer than 20 characters – truncated

Whenever a symbol name with more than 20 characters is defined, the name is truncated to 20 characters.

102 WARNING — Dirty programming style – Stack effects uncomputable

This warning occurs if you are using BRA instructions and lables within a MACRO/COLON definition. The compiler is not able to calculate the correct stack effects.

103 WARNING — Redefining a Number with a qFORTH word

By creating a new vocabulary entry and linking it in the word-list, this warning will occur if you have defined a new object with the name which already exists.

104 Undefined ROM–Segment

The name of the defined ROM segment is unknown.

105 Expected $ENDSEG – $ENDSEG inserted

If routines or ROM constants are to be placed into specific ROM segment blocks, they have to be enclosed by the compiler directives $BEGINSEG .... $ENDSEG. During compilation, the ROM segment directive $ENDSEG was expected because the compiler has found the beginning of an additional ROM segment definition. The directive $ENDSEG was inserted automatically.

106 No previous $BEGINSEG

The compiler found the directive $ENDSEG of a ROM segment definition without a ROM segment block being introduced by the directive $BEGINSEG.

107 Overriding previous Segment assignment

A routine within a $BEGINSEG .... $ENDSEG block overlapped with a SEGMENT directive for a single placement of a ROM constant or subroutine.

108 Expected SEGMENT – SEGMENT added

During compilation, the word SEGMENT was expected into the $DEFSEG directive. The compiler inserted the word SEGMENT automatically.

109 Defined Segment does not fit into parent segment

The defined ROM space area is greater than the ROM space area of the superior ROM segment block.

110 ROM–Segment of $AUTOSLEEP, $RESET, and INTx routines cannot be changed

The $AUTOSLEEP, $RESET and INTx routines have fixed addresses in the ROM base bank and can not move into another ROM segment.

153 Unexpected end of source file

The end of a source file was found before a definition in the source code was completed. Maybe a <CR> following a 'j'or ENDCODE statement is missing in the last line of a source file.

160 Only "@" or "!" operations are allowed with external objects

If you use external storage, the only operations on external objects, which have been declared as EXTERNAL before, are store and fetch operations. This means, for example, you can not push the address of an external object onto the stack and manipulate it.

161 Global labels in macros are not allowed

The use of global labels (beginning with _$) is forbidden, as this label would cause multiple definition problems when this macro is called.

163 $EXTMEMSIZE value is insufficient

The size of the external memory is too small, i.e., not all external objects can be placed.

164 You can not call that

The only thing that can be called by CALL-assembler-instructions are subroutines, defined by COLON definitions or labels within assembler subroutines.

165 Do not redefine assembler words

Assembler words can not be redefined.

166 Optimize XYTRACE : Not enough memory in partition list (****)

An internal list of the qFORTH compiler is too small.

167 Fatal error during XYTRACE (****)

A fatal internal compiler error occured during XYTRACE optimization. Please submit your source code to Atmel Wireless & Microcontrollers for failure analysis.

168 Partition–pointer–list too small (****)

An internal list of the qFORTH compiler is too small.

169 Invalid Context–Save/Context–Restore Macros

An internal compiler error occured during the compilation of a new system library.

170 Found operator where operand was expected

171 Found operand where operator was expected

172 String constant truncated to 80 characters

If a string constant with more than 80 characters is defined, the string constant is truncated after 80 characters.

197 Use $Bank_Switch FULL

Replace the default argument RESTRICTED of the compiler directive $BANK_SWITCH <arg> by the argument FULL.

198 Panic: Could not place subtree of SCALL–Routine in BB

To organize the base bank efficiently, the compiler has to place all routines with fixed ROM addresses into the zero page (INTx, $RESET, ...). The next step of the compiler is to place all routines and subroutines into the base bank which will be forced to a SCALL address. After that, the zero page is filled up with SCALL routines. This error occurs, if there is not enough memory space to place the corresponding routines into the base bank. You have to rearrange your source code .

199 TBL not actualized

Stack operation error caused by SWAP and DROP instruction. This failure may occur during compilation of a new system library.

200 Internal consistency check failed!

Internal compiler error! Please contact your Atmel Wireless & Microcontrollers sales person for immediate support.

201 WARNING — Different RET stack effect than in previous block

When processing a CASE .. OF .. ENDOF .. ENCASE statement, the current block has a different return stack effect than the previous block. This causes the stack effects of the current COLON/MACRO definition to be uncomputable.

202 WARNING — Different EXP–stack effect than in previous block

When processing a CASE .. OF .. ENDOF .. ENCASE statement, the current block has a different EXP stack effect than the previous block. This causes the stack effects of the current COLON/MACRO definition to be uncomputable.

203 Illegal word will be ignored

When processing a COLON/MACRO definition, an improper keyword was found (such as THEN without a prior IF or an UNTIL without a prior BEGIN ...) or within a ROMCONST definition an unusable word was

found. Within ROMCONST definitions, only numbers, constants, strings, and names of arrays, variables are allowed.

204 End of file reached while reading a string

When processing a string, the end of the source file was found before the end of the string was reached.

205 ',' expected

When defining a look-up table with ROMCONST, each item has to be followed by a blank and a comma, the last item too.

206 Index out of range

When indexing an array, the index was less than 0 or greater than the maximum index.

207 WARNING — Index expected

Everytime an opening square bracket after an array name is found, the compiler assumes to read an array index which has to be a number or a constant of the appropriate range.

208 WARNING — Value not in range 1 .. 16

The maximum index when defining a short byte or short nibble array has to be less than or equal to 15 and greater than or equal to 0. The index for any short array has to be a nibble. When defining an array, the bracketed number is the number of array elements whose index starts at 0 !

209 Value not in range 1 .. 256

The maximum index when defining a long byte or nibble array has to be less than or equal to 255 and greater than or equal to 0. The same range is valid when indexing a short array. The index for a long array has to be a byte.

210 WARNING — Unknown or invalid option

An unknown option was found when specifying an option list after a block within a COLON or MACRO definition.

211 WARNING — No INTERRUPT routine has been defined

Each program has to contain at least one definition of an interrupt service routine. The interrupt service routines are named INT0 to INT7. The compiler uses the defined interrupt routines to compute the set of used subroutines by following the execution path for every interrupt routine.

212 Recursion in CODE definitions not permitted

A macro definition can not be used in its own definition.

213 WARNING — Recursive stack effects unpredictable

The stack effects of a recursive COLON definition can not be computed. Nevertheless you can specify them in the option list using '[ExRy]'.

214 Inconsistent assembler definitions in system library (****)

This error indicates inconsistency in the definitions of assembler words in the system library.

215 WARNING — Forcing to an address overrides forcing to ZERO PAGE

If a COLON definition is forced to the zero page by the 'Z' option and forced to a specific address by AT, the AT overrides the 'Z'.

216 DPMI: General Protection Fault (****)

Internal compiler error! Please contact your Atmel Wireless & Microcontrollers sales person for immediate support.

217 Unable to place $RESET routine in ROM

The compiler was unable to place the $RESET routine at address 008h or there is not enough space to place the whole $RESET routine. Reduce the size of the $RESET routine.

218 Unable to place INTERRUPT routine in ROM

The compiler was unable to place an interrupt routine at a predefined address as there was not enough space to place the routine. The reason and the steps to avoid this are the same as described in error 217.

219 FATAL ERROR – Routine became longer when optimizing (****)

This means that a subroutine increased in length when it was optimized.

220 Insufficient ROM for placing ROM constant values

There was not enough space left in ROM when the compiler tried to place the ROM constant look-up table definition. The ROM constants are placed after all subroutines have been placed. If this error occurs, increase the size of the on-chip ROM (using $ROMSIZE) or break ROM constants and subroutines into smaller parts, so they can fit into smaller ROM holes not used.

221 FATAL ERROR during optimization (****)

A fatal error occured when optimizing the intermediate object code.

222 FATAL ERROR during ROM placement (****)

A fatal error occured when carrying out the ROM placement. Typical error if the ROM size is too small.

223 RET stack does not fit into RAM

The size of the return stack is greater than on-chip RAM. Increase the on-chip RAM or decrease the size of the return stack allocation.

224 EXP stack does not fit into RAM

The size of the EXP stack is greater than the size of on-chip RAM. Increase the on-chip RAM, decrease the size of the EXP stack or return stack allocation.

225 RET and EXP stack will overlap

You forced the EXP stack to a specific address so that both stacks do not use disjointed parts of RAM. Do not force the EXP stack to a specific address.

226 Not the name of a colon definition

You tried to use an assembler mnemonic or another reserved word of a colon definition.

227 WARNING — Unkown interrupt source of current routine – Stack effects 0,0 assumed

You have defined a SWI-instruction in the current routine, but the compiler cannot find out the interrupt which will activate this routine.

228 Insufficient $ROMSIZE for placing subroutines

Not all necessary subroutines could be placed in the MARC4's ROM. The actual ROM size is given to the compiler by the predefined constant $ROMSIZE. If this directive is not found in the source, a default ROM size is taken.

229 WARNING — RET stack size not set – Using default value

The return stack size was not set by the sequence VARIABLE R0 <xx> ALLOT where <xx> is the size in nibbles. The default value is 48 nibbles. The number of return stack entries is calculated by (<xx> /4) + 2.

230 WARNING — EXP–stack size not set – Using default value

The EXP stack size was not set by the sequence VARIABLE S0 <xx> ALLOT where <xx> is the size of the expression stack –1. The default value is 16 nibbles.

231 WARNING — Cannot determine target of SWI instruction

You have called an interrupt before the interrupt service routine is defined. Please try to rearrange your source code.

232 WARNING — Not all Z–optioned routines could be placed

There is not enough space in the zero page. Thus, not all subroutines, which are forced to a short-call address by the Z-option, could be placed on a short-call address. Decrease the length of interrupt or $RESET routines or use less forcing to a zero–page address by AT or Z-option.

233 WARNING — Using default value for $ROMSIZE

You did not set the predefined constant $ROMSIZE. The default value of 1.5K is taken.

234 WARNING — Using default value for $RAMSIZE

You did not set the predefined constant $RAMSIZE. So the default value of 111 nibbles is taken.

235 WARNING — Using default value for $EXTMEMSIZE

You defined external memory objects, but did not specify the size of the external memory. So a default value is taken. The default value is set to 255 nibbles. This warning is issued only if external objects have been defined.

236 WARNING — Using default value for $EXTMEMPORT

The predefined constant $EXTMEMPORT needs the port address for external memory accesses. The default value is Fh.

237 WARNING — Using default value for $EXTMEMTYPE

If external memory is used, the types RAM or EEPROM are valid parameters. RAM is the default parameter.

238 Unkown CRC–Algorithm. Valid are DEFAULT, SIMPLE and HARDWARE

The compiler supports three CRC algorithm which influence the checksum generation differently. Only DEFAULT, SIMPLE and HARDWARE are valid parameters. If there is no $CRC-directive defined in your source-code, the CRC algorithm is DEFAULT.

239 Unknown Bank switch method. Valid are Restricted and FULL

The compiler has found an invalid argument for the directive of the ROM bank switch. Only RESTRICTED and FULL are valid arguments. If there is no $BANK_SWITCH <arg> directive defined in your source-code, the $BANK_SWITCH argument is RESTRICTED.

241 No previous $IFDEF

When using conditional compilation, the compiler found a $ELSE or a $ENDIF but no previous $IFDEF.

242 Unmatched $IFDEF(s), possibly $ENDIF missing

When using conditional compilation, there were unmatched $IFDEF(s) left at the end of the source, i.e., there were more $IFDEFs than $ENDIFs.

244 Program aborted via <CTRL+C>

This error message is issued when the compiler is terminated via <CTRL+C> keyboard break.

245 – 247 TURBO–RT: TURBO runtime error (****)

This TURBO-runtime errors are caused by an incorrect compiler code. Please contact your Atmel Wireless & Microcontrollers' sales person for immediate support.

248 WARNING — Protect only assembler words in Colon or Code Definitions

The compiler directive $PROTECT must be used only to protect assembler words in colon or code definitions by creating a new library.

250 WARNING — $(NO)EXPAND meaningless in Colon definition

This compiler directive is only a macro expansion control.

251 TURBO–RT: TURBO runtime error (****)

This TURBO runtime error is caused by an incorrect compiler code. Please contact your Atmel Wireless & Microcontrollers' sales person for immediate support.

252 WARNING — Expansion mode has already been set globally

If the compiler directive $(NO)EXPAND is set on the outside of a code definition, the expansion mode is set globally. This global expansion mode can be set once in your source code only.

1.6.2 Uncoded Error and Warning Messages

Error message file <path> qFORTH2.MSG not found

The compiler's error message file is not available in the compiler's directory.

User library <name> not found

An user library was not found. Check, whether you supplied the correct extension, or if your user libraries have the default extension .LIB, whether you supplied the correct path or, if no path was supplied, whether they are stored in the same directory as the source file.

<name> is never called

You forced a routine to an address or to the zero page, which is never called directly by any other routine. You may omit this routine, or, if you do not force it, the compiler will not place it in ROM area.

ERROR — Illegal environment found by the compiler (****)

The compiler reads the program's environment to determine the path where the compiler overlay can be found. Use an MS-DOS operating system release 3.3 or above.

<list of files> : List is ignored

Do not pass more than one source file to be compiled. All except the first will be ignored.

<list of qualifiers> : List of qualifiers not allowed

The list of qualifiers <qualifix> is not allowed here and will be ignored.

Contradicting switch <switch> is ignored

You supplied two contradicting switches such as LIST and NOLIST. Omit one of them.

<qualified switch> : qualifying not allowed

You qualified a switch which does not allow qualifying.

<qualifix> : unknown qualifier

You specified an invalid qualifier after the /STATISTICS switch. Valid are only: NO, BRIEF, NORMAL and FULL (or abbreviations of these)

XY@!–optimizing requires XYLOAD–optimizing

You cannot optimize indirect–X/Y-fetch/store unless the loading of the X and Y register is optimized. XY@! optimizing is not completed.

XYTRACE optimizing requires XY@!–optimizing

You cannot carry out a register trace optimizing unless 'XY@!' is optimized. XYTRACE optimizing is not executed.

Unable to place <name> (<length> Bytes) (****)

The remaining space in ROM is too small to place the flagged object. Use less forcing to zero page or to an address or break large routines into smaller pieces. This warning will also produce a severe error.

This word is redefined (see <place of first definition>)

You defined the same object twice. This might cause problems when referenced by other routines. Rename or remove one object from your source code.

X@/Y@ in <name> : content could be changed by the optimizer

Within <name> you used the Y register, although XYTRACE optimization is done. This might change the normal content of the X or Y register. Don not use these registers by direct assembler instructions.

Call <routine> in <name> : Don't pass parameters in registers

<routine> uses the X or Y register. The compiler assumes that parameters to <routine> are passed in the registers. If XYTRACE optimization is carried out in <name>, the original content of these registers might change.

<name> can't be forced, because ROM address is fixed

Interrupt routines have fixed ROM addresses:

$AUTOSLEEP	000h	INT3	100h
$RESET	008h	INT4	140h
INT0	040h	INT5	180h
INT1	080h	INT6	1C0h
INT2	0C0h	INT7	1E0h

Unexpected end of file in library

When reading the system or a user library, an end of file was found before the library was read completely. Copy the qFORTH system library from the installation disk to your working directory. If the error still occurs, recompile your user library/libraries.

Illegal <record–type> record in library

The library file is destroyed by any reason or you are trying to read an old library with a new compiler version. Use the new system library, if you have a new compiler version, re-install the system library from the installation disk to your working directory. If the error still occurs, recompile your user-library/libraries.

<predefined value> is already set

A predefined value is set in more than one user library. This is not allowed. Recompile, so that the value is set in one user library only.

Duplicate label <labelname> in User Library

Two user libraries contain the same label name. Rename one of them.

<name> : name not found when reading cdrflist (****)

A part of your user library is lost by damaging the user library. Recompile it, if the error still occurs, send your files to Atmel Wireless & Microcontrollers for failure analysis.

No more room for intermediate code (****)

You have read in too many user libraries. Reduce the number of user libraries or split them up into a larger number and omit all libraries you do not need in this application program.

Your user library is inconsistent

You are trying to read an old user library in conjunction with a new version of the qFORTH system library. Recompile all your user libraries.

Switch <switch> is ambiguous

The switch <switch> is ambiguous, i.e., it is not possible to determine exactly which qualifier is meant (e.g., \LI is ambiguous (LIST or LIBRARY)).

Switch <switch> does not care me

You supplied an unknown switch.

Source file not found

The compiler did not find the specified source file. Maybe you specified an invalid path or the extension of your source file is not the default extension .SCR or .INC for an include file and you did not specify it.

System library QFORTH2.LIB not found

The compiler is looking for the system library in the same directory the compiler is stored in. Make sure that the system library is available as QFORTH2.LIB in that directory.

Qualifier <qualifix> is ambiguous

The qualifier <qualifix> in a qualifier list is ambiguous, i.e., it is not possible to determine exactly which qualifier you mean.

Qualifier <qualifix> does not care me

You supplied an unknown qualifier <qualifix> in the qualifier list. Indirect recursion not allowed. By re-definition of one or more objects you got an indirect recursion. This is not allowed.

<name> does not fit into ROM–hole

You forced a ROM item into a too small ROM hole by forcing another item near to this item in the ROM. Use less forcing or supply larger distances between the items.

<name> there is already something in ROM

You tried to force two items in the ROM. As result, they are overlapping. Move one of them to another location or use less forcing with AT.

<name> – defining occurence not found

An external labels was not defined. When an external label is used, its name has to be preceeded by '_$'. The name at the defining occurence must not be preceeded by these characters.

<predefined variable> — Value not set (****)

Indicates that a predefined variable was not set, not even with its default value.

<object> has not been defined

An external object (arguments of assembler instructions with operands or parts of a ROM constant) was not defined until the end of the source code.

<name> – out of address space (****)

A call of <name> or a branch to <name> exceeds of the maximum 4K address space

<array><index> : out of RAM space

An array fits only partially into RAM. Use less forcing with AT.

Conflict between RET–stack and <ram–object>

The RET stack and another RAM object do not occupy disjoint RAM. Use less forcing via AT or move one object.

Conflict between EXP–stack and <ram–object>

See above

Conflict between <ram–object> and an other RAM object

See above

<external object> is not in available external storage

An external object was forced via AT to a location outside the available external memory area.

Conflict between <name> and another external object

Two external objects do not occupy disjoint memory areas. Use less forcing via AT or move one object.