Compiler optimization levels and the debug view

4.3 Compiler optimization levels and the debug view

The precise optimizations performed by the compiler depend both on the level of optimization chosen, and whether you are optimizing for performance or code size.

The compiler supports the following optimization levels:

`O0`

Minimum optimization. Turns off most optimizations. When debugging is enabled, this option gives the best possible debug view because the structure of the generated code directly corresponds to the source code. All optimization that interferes with the debug view is disabled. In particular:

- Breakpoints can be set on any reachable point, including dead code.
- The value of a variable is available everywhere within its scope, except where it is uninitialized.
- Backtrace gives the stack of open function activations that is expected from reading the source.

Note

Although the debug view produced by `-O0` corresponds most closely to the source code, users might prefer the debug view produced by `-O1` because this improves the quality of the code without changing the fundamental structure.

Note

Dead code includes reachable code that has no effect on the result of the program, for example an assignment to a local variable that is never used. Unreachable code is specifically code that cannot be reached via any control flow path, for example code that immediately follows a return statement.

`1`

Restricted optimization. The compiler only performs optimizations that can be described by debug information. Removes unused inline functions and unused static functions. Turns off optimizations that seriously degrade the debug view. If used with `--debug`, this option gives a generally satisfactory debug view with good code density. The differences in the debug view from `-O0` are:

- Breakpoints cannot be set on dead code.
- Values of variables might not be available within their scope after they have been initialized. For example if their assigned location has been reused.
- Functions with no side-effects might be called out of sequence, or might be omitted if the result is not needed.
- Backtrace might not give the stack of open function activations that is expected from reading the source because of the presence of tailcalls.

The optimization level `-O1` produces good correspondence between source code and object code, especially when the source code contains no dead code. The generated code can be significantly smaller than the code at `-O0`, which can simplify analysis of the object code.

`2`

High optimization. If used with `--debug`, the debug view might be less satisfactory because the mapping of object code to source code is not always clear. The compiler might perform optimizations that cannot be described by debug information.
This is the default optimization level.
The differences in the debug view from `-O1` are:

- The source code to object code mapping might be many to one, because of the possibility of multiple source code locations mapping to one point of the file, and more aggressive instruction scheduling.
- Instruction scheduling is allowed to cross sequence points. This can lead to mismatches between the reported value of a variable at a particular point, and the value you might expect from reading the source code.
- The compiler automatically inlines functions.

`3`

Maximum optimization. When debugging is enabled, this option typically gives a poor debug view. ARM recommends debugging at lower optimization levels.
If you use `-O3` and `-Otime` together, the compiler performs extra optimizations that are more aggressive, such as:

- High-level scalar optimizations, including loop unrolling. This can give significant performance benefits at a small code size cost, but at the risk of a longer build time.
- More aggressive inlining and automatic inlining.

These optimizations effectively rewrite the input source code, resulting in object code with the lowest correspondence to source code and the worst debug view. The `--loop_optimization_level=option` controls the

amount of loop optimization performed at `-O3 -Otime`. The higher the amount of loop optimization the worse the correspondence between source and object code.
For extra information about the high level transformations performed on the source code at `-O3 -Otime` use the `--remarks` command-line option.

Because optimization affects the mapping of object code to source code, the choice of optimization level with `-Ospace` and `-Otime` generally impacts the debug view.
The option `-O0` is the best option to use if a simple debug view is required. Selecting `-O0` typically increases the size of the ELF image by 7 to 15%. To reduce the size of your debug tables, use the `--remove_unneeded_entities` option.

Related concepts

4.12 Benefits of reducing debug information in objects and libraries (armcc_chr1359124223955.htm)

Related reference

4.13 Methods of reducing debug information in objects and libraries (armcc_chr1359124224235.htm)
7.34 --debug, --no_debug (armcc_chr1359124909829.htm)
7.35 --debug_macros, --no_debug_macros (armcc_chr1359124910063.htm)
7.52 --dwarf2 (armcc_chr1359124915882.htm)
7.53 --dwarf3 (armcc_chr1359124916116.htm)
7.116 -Onum (armcc_chr1359124935804.htm)
7.121 -Ospace (armcc_chr1359124936615.htm)
7.122 -Otime (armcc_chr1359124936849.htm)
7.140 --remove_unneeded_entities, --no_remove_unneeded_entities (armcc_chr1359124942060.htm)

Related information

ELF for the ARM Architecture (http://infocenter.arm.com/help/topic/com.arm.doc.ihi0044-/index.html)