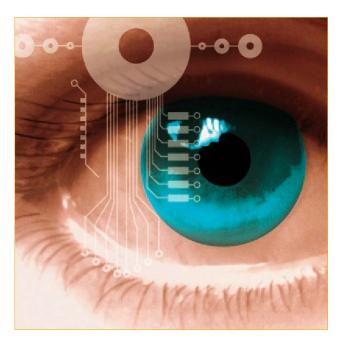
### **Benchmarking hints**

### How to tune IAR Embedded Workbench for best performance





Benchmarking hints

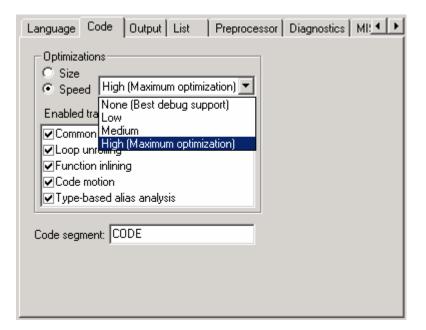
# **Optimization goal**

### Size or Speed?

Optimization level and type can be specified for the entire application and for individual files. In source code, the #pragma optimize directive allows you to do this even for individual functions.

The purpose of optimization is to reduce the code size and to improve the execution speed. When only one of these two goals can be satisfied, the compiler prioritizes according to the settings you specify.

Exploring the effects of the different transformations may lead to a better result. For example, the fact that *Function inlining* is more aggressive on speed optimization makes some programs smaller on the speed setting than on the size setting.





Benchmarking hints

# **Memory model**

#### Memory model

Choose the smallest possible memory model

#### Benefits:

Smaller addressesSmaller instructionsSmaller pointers

 $\Rightarrow$ More efficient  $\Rightarrow$ Less code

Target Output Library Configuration	Library Options   MISRA C
Processor variant © Core 1 © Core 2 © Core 3	
Code model Normal Position independent Large	Data model Tiny Small Large Use short address mode



Benchmarking hints

## **Runtime environment**

### Adapting the runtime environment

•By default, the runtime libraries are compiled at highest size optimization level. You should rebuild them if you are optimizing for speed!

•Select the required level of support for certain standard library functionality like, locale, file descriptors, and multibytes by choosing the appropriate library configuration.

•Select library options for scanf input and printf output formatters according to your needs. The smallest formatters are not selected by default.

Target Output Library	Configuration Library Options MISRA C
Library: Normal DLIB None Normal DLIB Full DLIB Custom DLIB	Description: Use the normal configuration of the C/EC++ runtime library. No locale interface, C locale, no file descriptor support, no multibytes in printf and scanf, and no hex floats in strtod.
Library file:	
\$TOOLKIT_DIR\$\LIB	di85-tnn.r85
Configuration file: \$TOOLKIT_DIR\$\LIB\	dl85-tnn.h

Target Output Library Configuration Library Options MISRA C
Printf formatter Tiny No specifier a or A, no specifier n, no float, no
flags.
Small
No specifier n, no float, no scan set, no assignment suppressing.



### **Data types**

# Data types have big influence on code size/speed

 Choose the smallest data types
 Use unsigned char if possible ⇒ Allows bit operations to be performed instead of arithmetic

Double floating-point size	1
32 bits	
C 64 bits	
	ľ

- <u>P</u> lain 'char' is	
◯ <u>S</u> igned	
• <u>U</u> nsigned	



Benchmarking hints

## **Target-specific options**

# Check for target-specific options that gain performance

#### Example:

Efficient addressing modes
 ⇒efficient memory accesses

 Locking registers for constants/variables
 ⇒more efficient code for operations on registers
 than on memory

 Even align functions entries
 ⇒even aligned instructions gain speed

 Byte align objects
 ⇒requires less memory for storage but might give
 bigger code

Use short address mode
Byte align objects       Short address work area         Word align function entries       Enable work area         20       Bytes
Use of registers Nr of locked registers: Two Put constants 255 and 65535 into registers Compatible with modules locking fewer registers
Number of registers to lock for global variables: 2 [R14R15]



Benchmarking hints

### **Benchmark code**

#### Use relevant benchmark code

•Embedded systems benchmarks shall address the characteristics of embedded programs.

•Real applications are usually good for benchmarks <u>but</u>, make sure that the code can execute. IAR XLINK will remove un-referenced code and variables but not all linkers have this ability.

•Make sure that the test code is not affected by the test harness (test support functions). The example to the right is actually benchmarking printf().

•Compare linked code. One compiler may inline code where another makes a library call.

•Use an application that you are familiar with!

```
unsigned long fib(unsigned long x)
{
    if (x > 2)
        printf("%ld\n",x); // Test harness
        return(fib(x-1)+fib(x-2));
    else
        return(1);
}
```



Benchmarking hints