1 Introduction

In the olden days, programming embedded devices meant programming directly in assembly language so that a programmer could scrutinize each instruction in order to optimize the program's performance as measured by execution speed or code size. However, today's modern compilers offer several optimization techniques for closing the gap between hand-written assembly language and compiled assembly language translated from a high-level language such as C. In this lab you will explore some of those optimization techniques and study their effects.

2 Lab Procedure

2.1 Research the Available Optimizations

1. The IAR Embedded Workbench IDE offers several optimizations as part of its compiler. To choose which optimizations are performed, you must navigate to the Optimizations Menu by clicking on Project → Options → C/C++ Compiler → Optimizations.
2. There you can select which optimization level you desire and customize which optimizations are performed.
3. The IAR C/C++ Development Guide for ARM (available on the lab website) documents the available optimizations (see Table 19: Compiler optimization levels on page 160).
4. Open the word document named *Description of Optimizations Table.doc* and fill in the table with a description of each optimization and their effects on execution time and code size. This document is included in the lab06.zip file.
5. To expedite this process, we have provided the table on the next page with links to reference material for each optimization technique.

Table 1: Links to Reference Material for Each Optimization Technique

Optimization Level	Optimization Description	References
None	Variables live through their entire scope	
Low	Variables only live for as long as they are needed	
Medium	Live-dead analysis and optimization	See Computers as Components p. 271
Medium	Dead code elimination	See Computers as Components p. 267 http://en.wikipedia.org/wiki/Dead_code_elimination
High	Redundant label elimination	http://tinyurl.com/redundant-label-elimination
High	Redundant branch elimination	http://tinyurl.com/Conditional-Branch-Elimination
High	Code hoisting	http://tinyurl.com/code-hoisting See Heiko Falk book p. 119
High	Peephole optimization	http://en.wikipedia.org/wiki/Peephole_optimization http://tinyurl.com/peephole-optimization
High	Some register content analysis and optimization	See Computers as Components p. 270 http://en.wikipedia.org/wiki/Register_allocation
High	Static Clustering	See IAR C/C++ Development Guide for ARM p. 164
High	Common subexpression elimination	See IAR C/C++ Development Guide for ARM p. 162 http://en.wikipedia.org/wiki/Common_subexpression_elimination
High	Instruction scheduling	See IAR C/C++ Development Guide for ARM p. 164 http://en.wikipedia.org/wiki/Instruction_scheduling
High	Cross jumping	http://tinyurl.com/Cross-Jumping
High	Advanced register content analysis and optimization	See Computers as Components p. 270 http://en.wikipedia.org/wiki/Register_allocation
High	Loop unrolling	See IAR C/C++ Development Guide for ARM p. 162 http://en.wikipedia.org/wiki/Loop_unwinding
High	Function inlining	See Computers as Components p. 267 See IAR C/C++ Development Guide for ARM p. 163 http://en.wikipedia.org/wiki/Inlining
High	Code motion	See IAR C/C++ Development Guide for ARM p. 163 http://en.wikipedia.org/wiki/Loop-invariant_code_motion
High	Type-based alias analysis	See IAR C/C++ Development Guide for ARM p. 163 http://en.wikipedia.org/wiki/Alias_analysis

References for compiler optimization in general:
http://www.compileroptimizations.com/category/

http://en.wikipedia.org/wiki/Compiler_optimization

2.2 Experiment with Optimizations

For each of the following optimizations generate source code that individually exercises the specific optimization. Then enable just that optimization from the Optimization Menu. To observe the effect of the optimization, open the main.list list file that is generated by the compiler. IAR Embedded Workbench stores list files in the following directory:
<lab06>/ECE3884 Labs\at91sam7l-stk\lab06\ewp\at91sam7l128_flash\List

You will need to save a copy of the list file generated with no optimizations to compare against. Show a TA your results for each optimization and save your source code that exercises each optimization. You will need to turn in your source code and an explanation of how you exercised each optimization. To generate a new list file, perform a "Rebuild All" (Project → Rebuild All).

1. Dead Code Elimination

   TA Initial Lab06 Box 1 ☐

2. Common Subexperession Elimination

   TA Initial Lab06 Box 2 ☐

3. Loop Unrolling

   TA Initial Lab06 Box 3 ☐

4. Function Inlining

   TA Initial Lab06 Box 4 ☐

5. Code Motion*

   TA Initial Lab06 Box 5 ☐

   *Note: To exercise code motion, you may also need to enable Common Subexpression Elimination.

3 Lab Report

Your lab report should contain the following items:

1. *Description of Optimizations Table.doc* filled in with descriptions of each optimization.
2. Source code that individually exercises each optimization from Section 2.2.
3. Description of how you exercised each optimization and the specific changes to the resulting assembly code.
4. Answer the following questions:
   What risks are associated with using compiler optimizations?
   What are some downsides of using compiler optimizations?