Investigating Compiler Technology

Introduction

Objectives

At the end of this lab you should be able to:

- Explain the stages of source code compilation
- Identify contents of a compiler symbol table
- Use selected optimization options of a compiler
- Analyse assembly code both in un-optimized and in optimized forms
- Describe different compiler optimizations methods

Basic Theory

Compilers are software engineering tools for generating executable binary code from high-level source code. The compilation process takes place in three stages

- Tokenising (or lexical analysis)
- Parsing (or syntax analysis)
- Code (assembly and/or binary) generation

Good compilers often produce highly optimised code in order to reduce the size of the binary code or speed up the execution of the code produced. As a result optimising compilers can directly contribute to CPU performance improvement.

Lab Exercises

The following exercises use the simulator's inbuilt compiler. Click on the **COMPILER…** button to show the compiler window.

1. Enter the following source code

```
program Ex1
    n = 5
    for i = 1 to 6
        n  = n + 1
        if n = 3 then
            n = 0
        end if
    next
end
```

Compile the above code by clicking on the **COMPILE…** button in the **SOURCE** frame.

Now observe the contents of the **COMPILER PROGRESS** textbox and try to answer the following questions:

What are the Pass 1, Pass 2 and Pass 3 stages called? Briefly explain the function of each stage based on your observations:

How many tokens are there in the above source code?

What do the indentations in Pass 2 stage signify?

What is the size of the code generated (code sizes are given in bytes)?

Click on the **SYMBOL TABLE…** button.

What do you observe in the displayed window? Briefly explain:

So, what is a symbol table and what sort of information does it normally contain?

In the **PROGRAM CODE** window, the numbers under the **LAdd** column are the Logical Addresses of the instructions, i.e. starting addresses in memory at which each instruction starts.

Suggest how these numbers are calculated by the compiler:

What do the numbers in the third column represent?

2. In the **COMPILER** frame click on the **Enable Optimizer** check box. Click on the **Redundant Code** check box in the **Optimizer** window. Compile the source again. Now observe the code generated.

Is it different than the previous one? Briefly comment:

What is the size of the code generated in this case?

What is the percentage change compared to the previous code size?

What explanation can you offer for this change?

3. Click on the **SHOW…** button in the **BINARY CODE** frame. You should see the **Binary Code for Ex1 window**.

Make a brief note of what you observe in this window:

Investigate the function of the **SHOW INSTRUCTION STATS…** button and make a brief comment below:

4. Use the **NEW** button of the source editor to open a new source editor tab. Enter the following source code

```
program Ex4
    n = 1 + 7 - 9
end
```

Make sure ONLY the **Enable Optimizer** AND the **Redundant Code** check boxes are checked. Now, compile this code and observe the code generated. Write down the code size:

Next check the **Constant Folding** check box. Compile the above code once again. What is the new code size?

How does the generated code differ from the previous code? Briefly explain. What is this optimization known as and how is it achieved?

5. Enter the following source code

```
program Ex5
    i = 3
    n = i * 16
end
```

Make sure ONLY the **Enable Optimizer** AND the **Redundant Code** check boxes are checked. Now, compile this code and note the code generated. Write down the code size:

Next check the **Strength Reduction** check box. Now, compile the above code once again. What is the new code size?

How does the generated code differ from the previous code? Briefly explain. What is this optimization known as and how is it achieved?

6. Enter the following source code

```
program Ex6
    for p = 1 to 8
        r = r + 2
    next
end
```

Make sure ONLY the **Enable Optimizer** AND the **Redundant Code** check boxes are checked. Now, compile this code and note the code generated. Write down the code size:

Calculate the number of instructions this program will execute when it is run and note it below:

Next check the **Loop Unrolling** check box. Now, compile the above code once again. What is the new code size?

How does the generated code differ from the previous code? Briefly explain. What is this optimization known as and how is it achieved?

Count the number of instructions this program will execute when it is run and make a note of it below:

Suggest how the optimisation is achieved in this case:

State one disadvantage of this optimization:

7. Review your observations of the four optimization methods studied above (see exercises 2, 4, 5 and 6). Using this information identify which of the **I** (i.e. total number of instructions executed) and **CPI** (i.e. clocks per instruction) parameters of the CPU performance equation is impacted by each of the following methods:

	I	**CPI**
Redundant code optimization		
Constant folding optimization		
Strength reduction optimization		
Loop unrolling optimization		

8. We studied a very small selection of compiler optimizations. Research and identify additional three popular compiler optimizations and make a very brief note of each below: