Investigating Compiler Technology

Introduction

Objectives

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

- Understand the stages of source code compilation
- Use selected optimization options of a compiler
- Analyse generated assembly code both in un-optimized and in optimized forms
- Explain the functions of different compiler optimizations

Processor (CPU) Simulators

The computer architecture tutorials are supported by simulators, which are created to underpin theoretical concepts normally covered during the lectures. The simulators provide visual and animated representation of mechanisms involved and enable the students to observe the hidden inner workings of systems, which would be difficult or impossible to do otherwise. The added advantage of using simulators is that they allow the students to experiment and explore different technological aspects of systems without having to install and configure the real systems.

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

- Parsing or tokenising
- Language 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.

Simulator Details

This section includes some basic information on the simulator, which should enable the students to use the simulator. The tutor(s) will be available to help anyone experiencing difficulty in using the simulator.

The simulator for this lab is an application running on a PC and is composed of multiple windows.

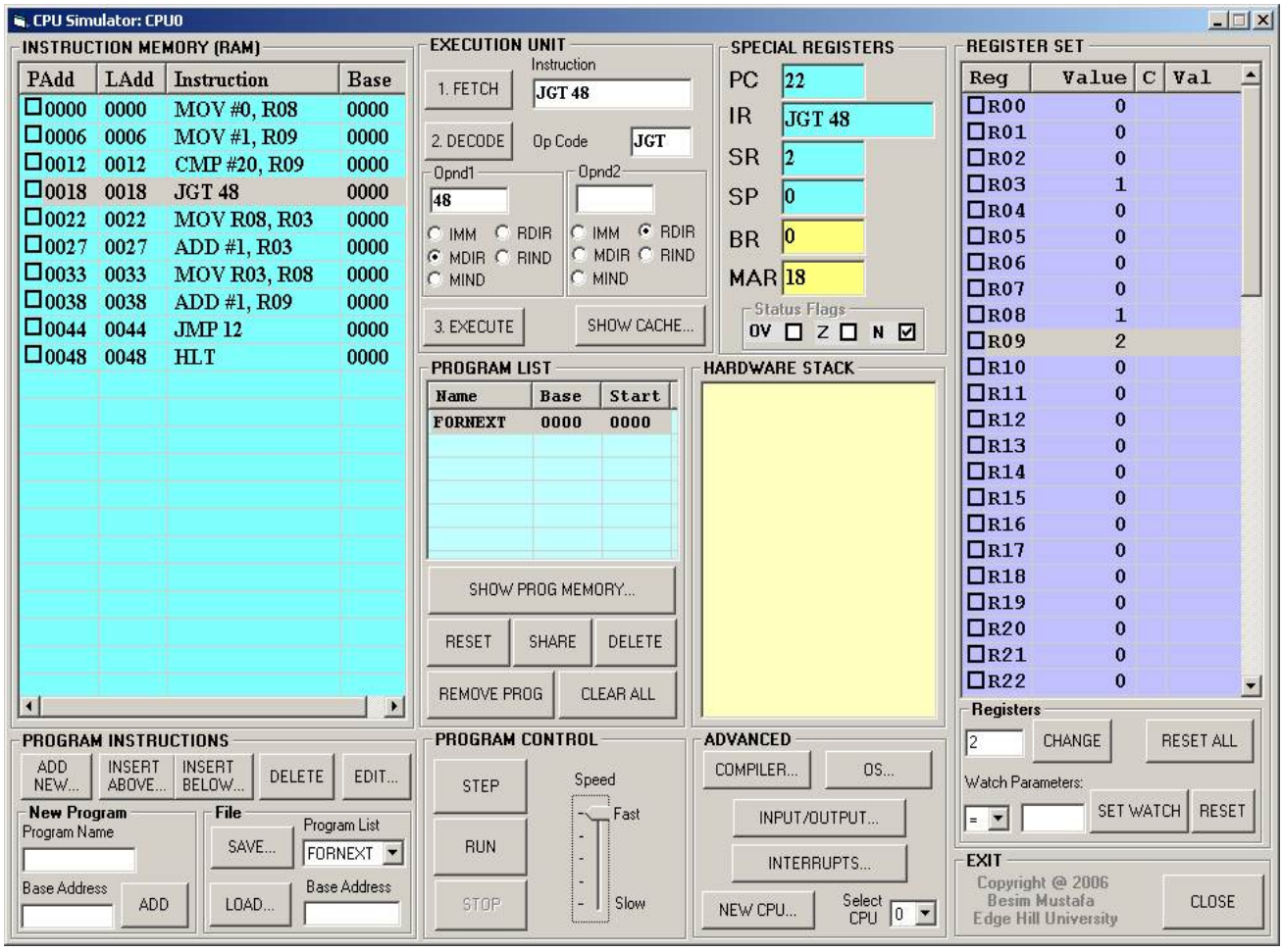

Image 1 - Main simulator window

The main window shown in Image 1 is composed of several sub-views, which represent different functional parts of the simulated processor. For this lab session we are interested only in the compiler part of the simulator.

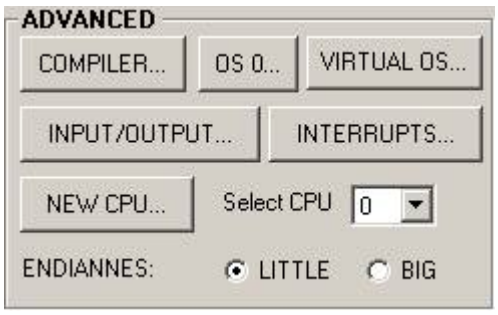

Image 2 - Advanced functions

In order to access the compiler, click on the **COMPILER…** button as shown in Image 2 on the right. The compiler window shown in Image 3 below will show.

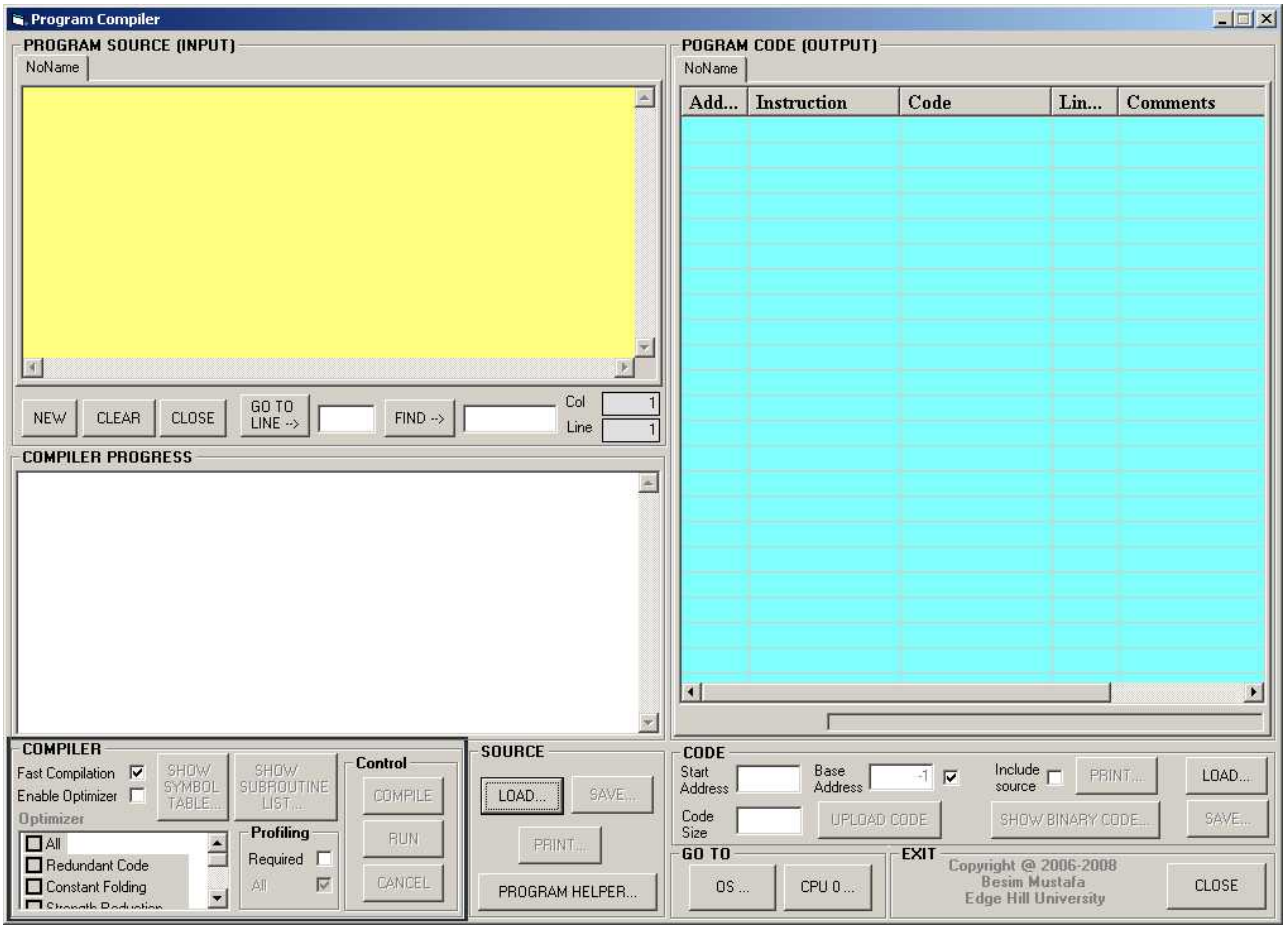

Image 3 - The main compiler window

In the compiler window there are three main sub-windows

- **Program Source** - all high-level source statements appear here.
- **Compiler Progress** - information on the progress of a compilation appear here.
- **Program Code** - assembly code generated by the compiler appear here.

Lab Exercises - Investigate and Explore

The lab exercises are a series of experiments, which are attempted by the students under guidelines. The students are expected to carry out further investigations on their own in order to form a better understanding of the technology.

Now, have a go at the following activities:

1. Enter the following source code

```
program Test
    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. Make a note of the kind of information displayed here and try to answer the following questions:

- What are Pass 1, Pass 2 and Pass 3 stages?
- How many tokens are there in the above source?
- What do the indentations in Pass 2 and Pass 3 stages indicate?

Click on the **SHOW SYMBOL TABLE…** button. What do you observe? Explain. What is a symbol table? Double-click on a symbol line in this window and observe what happens in the **PROGRAM SOURCE** window

- What is the size of the code generated?
- In the **PROGRAM CODE** window, what do the numbers on the left indicate?
- Suggest how these numbers are calculated?
- What do numbers in the third and the fourth columns indicate?

Try the following:

Click on the second line (i.e. **`n = 5`**) in the source code and observe what happens in the **PROGRAM CODE** window. Now, click the down arrow repeatedly on the keyboard and again observe what happens in the code window. What are your comments?

Now, try this:

Click on the first line assembly code in the code window and observe what happens in the source window. Now, repeatedly click the down arrow and observe what happens in the source window. What are your comments?

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?
- What is the size of the code generated in this case?
- What is the percentage change?
- What explanation can you offer for the change?

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

- What do you observe in this window?
- What is the function of the **SHOW INSTRUCTION STATS…** button? Investigate.
- Click on **NEXT INSTRUCTION** button and repeatedly click on the → button until the **SKIP TO NEXT** button is enabled. Now, repeatedly click on the **SKIP TO NEXT** button. What do you observe? Explain.

4. Replace the above source with the following source code (Alternatively you can use the **NEW** button to open a new source tab).

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

Make sure the **Enable Optimizer** check box is NOT checked**,** i.e. the compiler optimiser is disabled. Now, compile this code and note the code generated. Write down the code size.

Check the **Enable Optimizer** check box, i.e. the compiler optimiser is enabled. Also check only the **Redundant Code** check box AND the **Constant Folding** check box. Now, compile the above code once again.

- What is the new code size?
- How does the generated code differ from the previous (un-optimised) code? Explain.

5. Enter the following code

```
program Test
    n = 3
    n = n * 16
end
```

Make sure the **Enable Optimizer** check box is NOT checked**,** i.e. the compiler optimiser is disabled. Now, compile this code and note the code generated. Write down the code size.

Check the **Enable Optimizer** check box, i.e. the compiler optimiser is enabled. Also check only the **Redundant Code** check box AND 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 (un-optimised) code? Explain.

6. Enter the following code

```
program Test
    for p = 1 to 9
        r = 0
    next
end
```

Make sure the **Enable Optimizer** check box is NOT checked**,** i.e. the compiler optimiser is disabled. Now, compile this code and note the code generated. Write down the code size.

- Count the number of instructions this program will execute when it is run

Check the **Enable Optimizer** check box, i.e. the compiler optimiser is enabled. Also check only the **Redundant Code** check box AND 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 (un-optimised) code? Explain.
- Count the number of instructions this program will execute when it is run
- Suggest how the optimisation is achieved in this case.