Lab Experience 17

Programming Language Translation

Objectives

- Gain insight into the translation process for converting one virtual machine to another
- See the process by which an assembler translates assembly language into machine language
- See some of the major steps of the process a compiler would use to translate programs using a small subset of Java into assembly language

Background

In previous lab experiences, you observed programs in three very different programming languages. You began with a language whose form maps directly to the physical structure of a real computer. You expressed instructions and data in terms of strings of 1s and 0s in this language. The computer easily understood what these strings meant, but you probably had to go through a painful process of translation to figure out what you were saying in machine language. Then you learned to use a language that still reflects the structure of the computer, but employs terms that are closer to those you would use if you were describing the instructions and data to another person. This language requires a program called an assembler to translate programs to machine language. Finally, you learned to use a language whose sentences more closely resemble those you would use in English to describe the solution of a problem. This language, Java, requires a program called a compiler to translate your sentences into machine language.

The main point of these lab exercises was not only to show you that there are different languages for expressing programs. The languages you used also represent layers of virtual machine bridging the gap between the structure of a real computer and the structure of your thoughts. It is no exaggeration to say that if this gap had not been bridged, most of the interesting and useful software that we have would not exist.

The disciplines of programming language design and translation are concerned with constructing languages and translators that help to close the conceptual gap between computers and human beings. The purpose of the present lab experience is to expose you to the translation process for converting one virtual machine into another. The following lab exercises will allow you to explore what happens when a program translates code in one language into code in another language. First you will examine how an
assembler translates an assembly language program to a machine language program. Then you will study the process by which a Java compiler can translate a Java program to an assembly language program.

One note of clarification is in order before we begin this lab. We said that a Java compiler "can" translate Java programs to assembly language programs, but most Java compilers actually translate Java programs to programs whose code is called byte code. Byte code programs are then executed by a byte code interpreter, sometimes also called a Java virtual machine (JVM). A byte code interpreter works something like the LISP interpreter mentioned in Lab 16, except that the Java programmer never sees the byte code. However, for the purposes of the current lab, we use a Java compiler for a very small subset of Java that does translate Java programs to assembly language programs.

For exercises 17.1-17.4, select the Assembler from the main menu.

Exercise 17.1 Syntax analysis

Open the file Example1.asm from the File menu of the assembler. You should see the assembly language program for adding two numbers in the source pane. Select Assemble from the Assembler menu. You should see a program listing appear in the listing pane, with no syntax errors reported. Before a program can be translated, the translator must verify that there are no syntax errors in the source program. Now go back to the source pane and edit the program by changing .begin to begin (just delete the period from that line). Run the assembler again and watch what happens. Do you see where the error occurred, and was the message about it informative? Now put the period back in front of begin and edit the source program again, by deleting the t from halt. What kind of message do you suppose you will see this time? Why do you think that the assembler bothers to signal that there are syntax errors and to require that programs be syntactically correct before allowing the process to continue?

Exercise 17.2 Static semantic analysis

Static semantic analysis is the part of the translation process that obtains information about the meanings of identifiers (variables, constants, types, and functions) at compile time. Reload the example assembly language program, and edit it by replacing add y with add a. Assemble the program and think about the error message. Suppose that the assembler ignored errors of this sort. What do you think would happen when the computer tries to execute the instruction add a? Now reload the program, and insert the instruction jump q right after .begin. Assemble the program and explain the error. Insert a label for q before halt and reassemble. Then insert the label k before add y and reassemble. What are the rules governing instruction labels in assembly language, and how are they different from variable labels? Now reload the program, and change the number 4 to 32768 and reassemble. What sort of error is this, and why is it important to detect it?

Exercise 17.3 Object code

Reload the example assembly language program, assemble it, and select View Object Code from the Assembler menu. Is there a one-to-one correspondence between each assembly language instruction or datum and each machine code instruction or datum? Is there a direct mapping between the sentences of assembly language and the sentences of machine language? Verify from the table of opcodes in Lab Experience 9 that this is so. Now pick Execute from the Assembler menu. Step through the execution of the program by the machine language interpreter. Try to describe how machine language and assembly
language are similar. Try to describe what is involved in translating a single assembly language instruction to a machine code instruction (picking an instruction from the example program will help).

**Exercise 17.4 Symbol table**

Return to the assembler from the machine language interpreter, and pick **View Symbol Table** from the **Assembler** menu. Verify that the address of each data label is correct. Why are there no instruction labels in the table? Add an instruction label to the source program, reassemble, and view the symbol table once more. Is the address of the instruction label correct? How do you suppose that the assembler uses the symbol table to detect errors in the source program?

For the following exercises, select the **Language Translation** from the main menu. Then select **Java** from the **Language** menu.

### 17.5 Syntax analysis

Open the program **Example1.java** from the **File** menu. You should see a Java program displaying the sum of two numbers in the source pane. Pick **Compile** from the **Translator** menu. This is the only Java program in the Examples folder that compiles correctly in this lab. The reason for this is that the subset of Java used for this lab is much smaller than the one used in labs 12-14. In particular, the present version of Java lacks the data types **double**, **char**, string, and array, the operators *, /, and %, and you cannot define methods. The reason for this is that these features do not translate to features supported by our assembly language.

Now try introducing the following syntax errors into the program, making sure that you reload the program from the file for each experiment:

a. Delete the word **main**, and compile.

b. Delete the first {, and compile.

c. Delete the first assignment operator (=), and compile.

d. Delete the addition operator (+), and compile.

How informative are the syntax error messages of this compiler, as compared with those generated by the assembler? Why is the compiler so fussy about Java's syntax?

**Exercise 17.6 Static semantic analysis**

Change the word **sum** to the word **result** in the list of variables declared at the top of the program, and compile it. What information do the error messages provide? Why do you suppose that Java identifiers must be declared before they are used in statements? Now change the number 4 to a large number, such as 32768. Compile the program and explain why it is useful that the compiler catches this error before the program is translated and executed. Also explain why the compiler allows the number 32767.
Exercise 17.7 Object code

Reload and compile the example program. Select View Object Code from the Translator menu. You should see the equivalent program in assembly language in the Object Code pane (Figure 17.1).

![Java Translator - Example1.java](image)

Figure 17.1 The object code of a Java program

Look at the data declaration part of the assembly language program, and compare that with the data declarations in the Java program. Note that there is a correspondence between the Java data names and the data labels in the assembly language program. How many assembly language instructions, on the average, are required to implement a Java statement? Which Java statements are the easiest to translate, and which ones seem to be the most difficult?

Exercise 17.8 Translating a Java loop

Edit the program so that it displays all of the numbers between the first number and the second number. Compile the program and view the object code. You should see an equivalent loop in assembly language.

Exercise 17.9 Code optimization

Examine the translation of the Java assignment statement `first = first + 1` in the assembly language program. How many instructions does this simple Java statement require? Explain how the compiler could have translated this Java assignment statement to one assembly language instruction (Hint: change the Java statement to `++first` and recompile). Explain how the translated program would run more efficiently if the compiler as a rule translated such statements this new way.

Exercise 17.10 Symbol table

Pick View Symbol Table from the Translator menu. Explain the difference between a Java
constant and a Java variable. Try putting a Java constant on the left side of an assignment statement in the Java program, and compile the program and explain the error message. How do you suppose that the compiler uses the symbol table to translate the Java program to an assembly language program? Explain the differences between a Java compiler's symbol table and an assembler's symbol table.
Worksheet
Lab Experience 17
Programming language Translation

Name: ________________________________

Course: ______________________________

Exercise 17.1 Syntax analysis

Exercise 17.2 Static semantic analysis

Exercise 17.3 Object code
Exercise 17.4 Symbol table

17.5 Syntax analysis

a. Delete the word main, and compile.

b. Delete the first {, and compile.

c. Delete the first assignment operator (=), and compile.

d. Delete the addition operator (+), and compile.

Exercise 17.6 Static semantic analysis
Exercise 17.7 Object code

Exercise 17.9 Code optimization
Exercise 17.10 Symbol table