CSC 345 Lab – Assembly Language Programming

Overview

Practice writing pseudo-assembler programs. Write your assembly language programs in a text file. To run your assembly programs, you will need to write a Java program that uses the assembly language library that you will be given (download the library from Brightspace). Your Java program will need to read an assembly language program from a text file and then pass it to the assembly language library for processing and running.

Part 1

Create an IntelliJ console application. Make sure to use Maven as the build system when creating the project.

Part 2

Add the PseudoAssemblyObf library to the IntelliJ project and run a test assembly program.

- Add the PseudoAssemblyObf library to your project.
    - Download the PseudoAssemblyObf library from Brightspace (if necessary).
    - Install the PseudoAssemblyObf library in the local Maven repository (if necessary).
    - Add the PseudoAssemblyObf library as a Maven dependency to your IntelliJ project. Here is the dependency code:

```
<dependencies>
  <dependency>
    <groupId>org.example</groupId>
    <artifactId>PseudoAssemblyObf</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>
```

- Create a new text file named "assemPart2.txt". Add the following assembly language code to the file:

```
.data
.code
loadintliteral ri1, 88
printi ri1
```

- Run the program. You should see 88 as the output.

Part 3

Use the Program class in the assembly library to parse and execute your assembler programs. When creating an instance of Program use ".\\target\\classes" for the output directory parameter of the Program class constructor. If you do not use this directory, you may receive a ClassNotFoundException when you run the program.

IMPORTANT HINT: There is sample Java code to setup the Program class and reading assembly language code from a file at the end of this document (see below).

Write an assembler program that does the following:

- Store the constant number 77 in register ri1.
- Store the constant number 11 in register ri2.
- Add the values in registers ri1 and ri2 and put the sum in register ri3.
- Increment the value in registers ri1, ri2, and ri3.
- Print the values of registers ri1, ri2, and ri3 on screen.

Part 4

Write a program to multiply two variables with each other. Some items below will require multiple instructions.

- Declare three variables named a, b, and c.
- Put the value 10 in a.
- Put the value 20 in b.
- Print the values of variables a, b, and c on screen.
- Write code to multiply the values in variables a and b and store the result in variable c.
    - Do not assume that values you need are already in registers. This means you will have to load the variables a and b into registers.
    - The mult instruction only works on registers (not variables).
- Print the values of variables a, b, and c on screen again.

Part 5

Write a program that compares if variables are equal and writes a message.

- Put the value 20 in register ri1.
- Put the value 30 in register ri2.
- Test if the values in ri1 and ri2 are equal. If they are equal then print the message "Values are equal". If they are not equal then print the message "Values are not equal".
- Print the message "Program done" on screen at the end of the program.
- Run the program and make sure the correct output appears.
- Update the code so that it puts the value 20 in ri2.
- Run the program and make sure the correct output appears.

Part 6

Write a program that displays the numbers 1 through 10 on screen. Your program should have a "loop" that displays each number that is printed. Hint: Use a branching instruction to help create the loop.

Part 7

Write a program that adds the numbers 1 to 5 and prints the result on screen. Your program should use a “loop” to add the numbers. Hint: Use a branching instruction to help create the loop.

Sample Java Code for the Program Class and Reading Code from a File

```
String code = "";

try {
   FileReader fr = new FileReader("assemPart2.txt");
   Scanner infile = new Scanner(fr);
   while (infile.hasNext()) {
      code += infile.nextLine();
      code += "\n";
   }
} catch (FileNotFoundException e) {
   throw new RuntimeException(e);
}

int numVirtualRegistersInt = 32;
int numVirtualRegistersString = 32;
String outputClassName = "MyProgram1";
String outputPackageNameDot = "mypackage";
String classRootDir = System.getProperty("user.dir") + "/" + "target/classes";
PseudoAssemblyWithStringProgram pseudoAssemblyWithStringProgram = new
      PseudoAssemblyWithStringProgram(
      code,
      outputClassName,
      outputPackageNameDot,
      classRootDir,
      numVirtualRegistersInt,
      numVirtualRegistersString
);
boolean parseSuccessful;
parseSuccessful = pseudoAssemblyWithStringProgram.parse();
if (parseSuccessful == true) {
   // Creates a Java bytecode class file
   pseudoAssemblyWithStringProgram.generateBytecode();
   // Run the Java bytecode class file and show output on the console
   PrintStream outstream = new PrintStream(System.out);
   pseudoAssemblyWithStringProgram.run(outstream);
} else {
   String messages = pseudoAssemblyWithStringProgram.getAllParseMessages();
   System.out.println(messages);
}
```