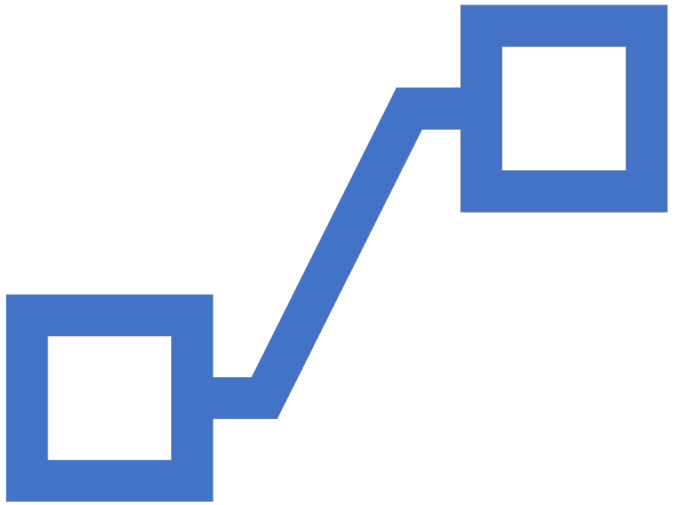
CSC 416/565: DESIGN AND CONSTRUCTION OF COMPILERS

West Chester University
Dr. Richard Burns
Fall 2023



Code Generation

Programming Assignment #5
Introduction

Role of These Slides

- These slides are to *supplement* the `README.md` project description in the GH repo starter code, as well as the lectures
- Consult all resources and ask questions on discord
- See these slides for a visual presentation of some of the subtleties in the assignment

Overall Task

- You'll be implementing the **code generation** back-end into your compiler
  - We haven't talked about **optimization** yet, so after covering that topic we could (in a longer semester) come back to this programming assignment and incorporate that in
- This project involves:
  1. Running `.ram` programs through your front-end
     - Recall that the last phase of the front-end is **semantic analysis**, which builds an AST that you can traverse using visitors. This phase also constructed a symbol table for your `.ram` source program.
  2. Emit semantically and syntactically correct assembly instructions for your `.ram` program
- The `README.md` lists tasks in more detail

Project Structure

There are some new files in this starter code, including:

1. A complete front-end for the Ram23 grammar specification
   - … so if you didn't finish Programming Assignments #4, everyone is starting from the same spot for this assignment
   - I will also not test your code on `.ram` programs that should generate front-end compile time or back-end run time errors
2. An updated driver (`compilers.Ram23Compiler`)
3. Starter code for your code generation visitor (`visitors.CodeGenerator`)
4. Test programs (`test/java/compilers/test_programs`)
5. I also bundled the `MARS4_5.jar` MIPS simulator into a `lib` folder in the project (not the best SE solution, but it saves you a couple of `mvn` commands that you would otherwise have needed to perform)

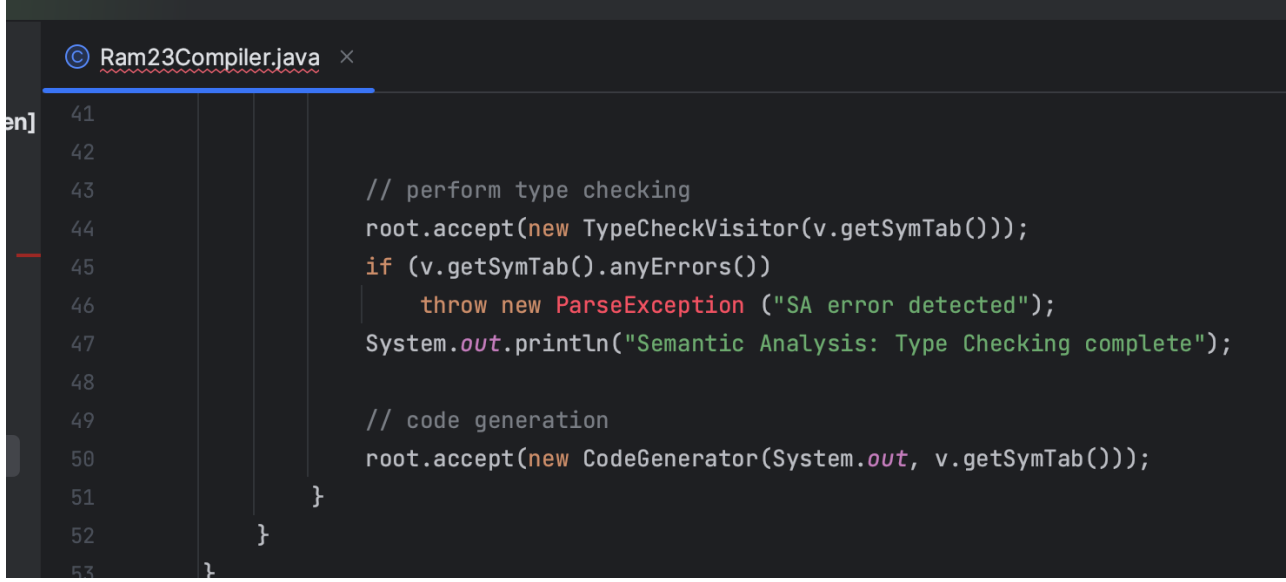
Testing Programs

- In the `test_programs` directory, there are many `.ram` programs that are referenced in the `README.md`
- Also included are `.correct` outputs that would be the executed output if the `.ram` programs are run
- Your task is to have your code generator automatically create `.s` assembly code for these `.ram` programs
  - (Assembly code is often saved with the extension `.asm` or `.s` or `.a`)
- The testing framework runs the MIPS simulator MARS on your code to assemble and run your generated `.s` file; output is saved in a `.s.output` file. If this output matches `.correct`, it will pass the test case

Code you will be modifying

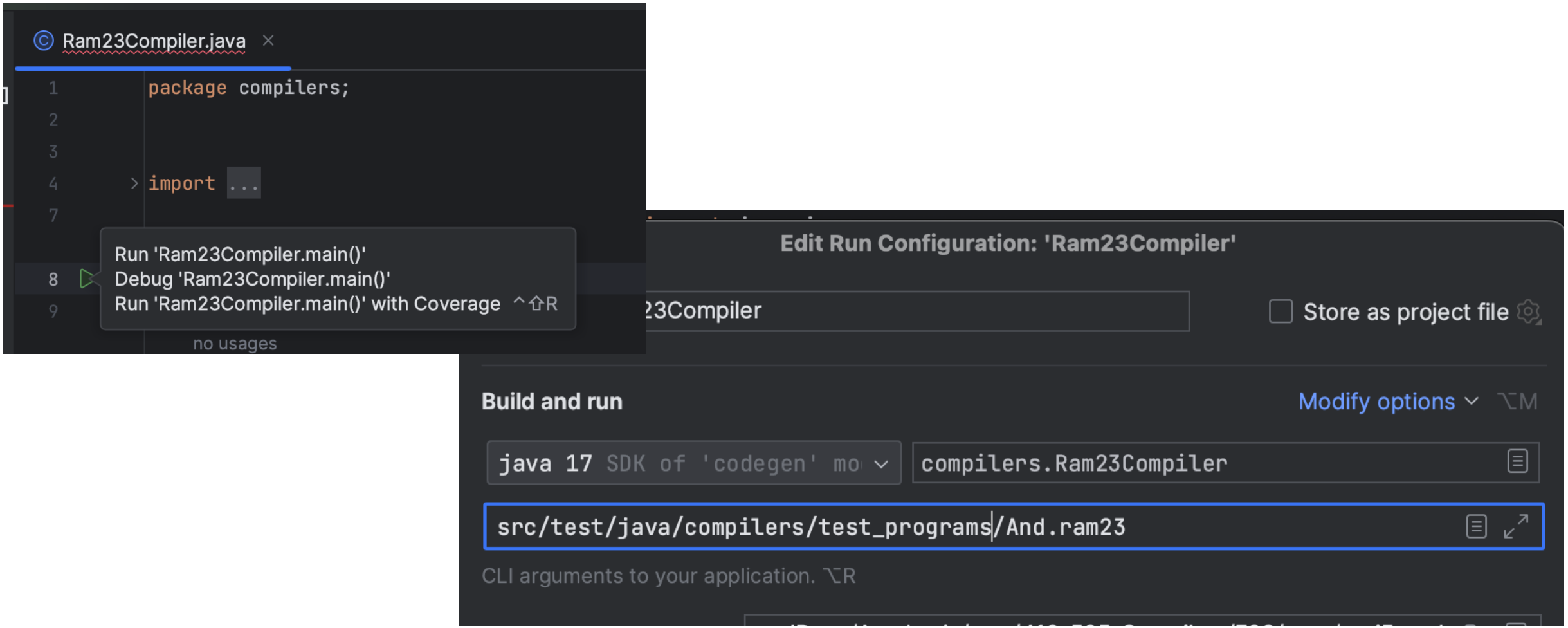
- 99% of your new code will be in `visitor/CodeGenerator.java`
- When you get to Task #8 in the `README.md`, you might also make a very minor addition to `symboltable/RamVariable.java`

An initial experiment



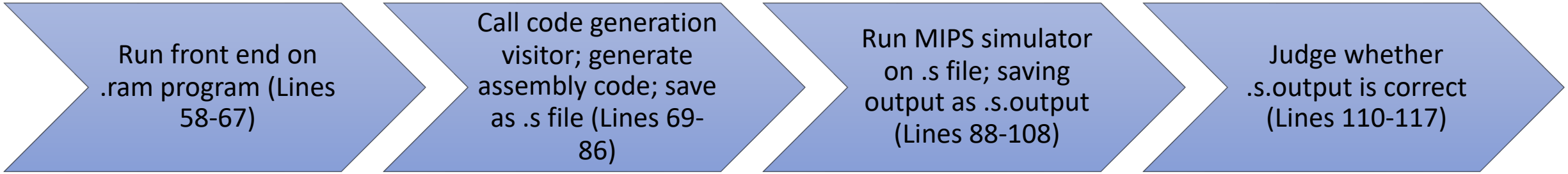
- Perform a `mvn clean`, and `javacc:jaccc` to generate the `RamParser.class`
- Also perform a `mvn compile`
- See `src/main/java/compilers/Ram23Compiler.java`
- Notice that the `Ram23Compiler` is now extended on a Line 50 call to a visitor that performs the code generation once the SA task is finished
- The code generation visitor takes two arguments:
  1. Where to emit/print the assembly code
  2. The symbol table that was created during the SA task

Remember to test your code on single files using the driver



Test Suites

- Performing a `mvn test` will run all uncommented `.ram` programs in `test/java/compilers/CodegenTest.java`
- The big method used for testing here is `testRamProgram()` which times out after 20 seconds
- You do not have to edit `testRamProgram()`! This is what is going on there:



Compute

- Running all of the tests takes a good bit of compute as you'll see.
- I recommend testing your visitor on *individual files* using the driver until the very end!

- Before starting, make sure that you re-read Appendix A and our course references, and that you are on the way to being comfortable with assembly and MIPS

Things to watch out for…

- If the test suite fails/crashes for a program with an error message that `.s.output` is empty or cannot be read, the most likely cause is that your visitor's generated `.s` file has an issue.
- Try opening this `.s` file manually in the MIPS and *assemble* it
  - Were you able to load the file? Were there any flagged syntactic errors?
  - Try running the file. Did it generate output and successfully terminate?