CS4120/4121/5120/5121—Spring 2016

Programming Assignment 5

Assembly Code Generation

Due: Wednesday, April 13, 11:59PM

This programming assignment requires you to implement an *assembly-code generator* for the Xi programming language. Assembly code is generated from the intermediate representation, making your compiler fully functional. The assembly code should be processable by the GNU assembler and linkable with the runtime library we provide in order to produce working executables.

0 Changes

- None yet; watch this space.

1 Instructions

1.1 Grading

Solutions will be graded on documentation and design, completeness of the implementation, correctness, and style. 5% of the score is allocated to whether bugs in past assignments have been fixed.

1.2 Partners

You will work in a group of 3–4 students for this assignment. This should be the same group as in the last assignment. If not, please discuss with the course staff.

Remember that the course staff is happy to help with problems you run into. For help, read all Piazza posts and ask questions (that have not already been addressed), attend office hours, or meet with any course staff member either at the prearranged office hour time or at a mutually satisfactory time you arrange.

1.3 Package names

Please ensure that all Java code you submit is contained within a package whose name contains the NetID of at least one of your group members. Subpackages under this package are allowed; they can be named however you would like.

1.4 Tips

You should complete your implementation of assembly-code generator as you see fit, but we offer the following suggestions.

First, download and compile the runtime. Take a look at the `.s` files inside the `examples` directory, and try assembling and linking them by hand. If you can do it for the examples, you will be able to run programs your compiler produces.

As part of your implementation, you will be specifying many different tiles and their mapping from the IR to the assembly code. Plan out how these tiles will be specified and organized.

Since you are now able to produce runnable programs, you can have runnable functionality test cases. Taking advantage of that can help automate testing. However, do not limit yourself to such tests, as it may be hard to get good coverage over instruction-selection cases from `.xi` sources alone.

2 Design overview document

We expect your group to submit an overview document. The Overview Document Specification outlines our expectations.

3 Building on previous programming assignments

Use your lexer from PA1, your parser from PA2, your type checker from PA3, and your IR generator from PA4. Part of your task for this assignment is to fix any problems that you had in the previous assignments. Discuss these problems in your overview document, and explain briefly how you fixed them.

4 Version control

As in the last assignment, you must submit file `pa5.log` that lists the commit history from your group since your last submission.

5 Runtime library

We require the code you produce to be able to interface with the runtime we provide, and to interoperate with other functions we may create for testing. To do this, it is sufficient to follow the ABI specification. Most of the required work was likely done in the previous assignment. For this assignment, you should finish implementing the details that were still abstract at the IR level. In particular, you will need to be aware of the rules on caller- and callee-saved registers.

6 Quality of assembly code

We do not expect you to implement optimizations or high-quality register allocation for this assignment; the goal here is to produce working programs. We do expect you, however, to implement nontrivial *instruction selection* with tiles that take advantage of the expressive x86-64 instruction set features like complicated addressing modes and in-memory operands. You are free to implement register allocation if you wish, but it is fully acceptable to stack-allocate all temporaries and use registers only to shuttle data between instructions and the stack.

7 Calling conventions and binary compatibility

We require that you support the System V ABI so that the course staff can run your submissions on the Linux VM. See Section 4 of the ABI specification for details.

8 Provided code

We are providing a few test cases for preliminary testing of your compiler. These test cases are not to be considered exhaustive. You will need to develop your own test cases to properly test your compiler.

9 External links

The following manuals may be useful:

- Intel® 64 and IA-32 Architectures Software Developer Manuals
- GNU Assembler manual

10 Command-line interface

A general form for the command-line interface is as follows:

```
xic [options] <source files>
```

Unless noted below, the expected behaviors of previously available options are as defined in the previous assignment. `xic` should support any reasonable combination of options. For this assignment, the following options are possible:

- `--help`: Print a synopsis of options.
- `--lex`: Generate output from lexical analysis.
- `--parse`: Generate output from syntactic analysis.
- `--typecheck`: Generate output from semantic analysis.
- `--irgen`: Generate intermediate code.
- `--irrun`: Generate and interpret intermediate code (optional).
- `-sourcepath <path>`: Specify where to find input source files.
- `-libpath <path>`: Specify where to find library interface files.
- `-D <path>`: Specify where to place generated diagnostic files.
- `-d <path>`: Specify where to place generated assembly output files.

  For each source file given as `path/to/file.xi` in the command line, an output file named `path/to/file.s` is generated to contain the assembly output of the source file. If `path` is given, the compiler should place generated assembly output files in the directory relative to this path. The default is the current directory in which `xic` is run.

For example, if this path is `o/u/t` and the file to be generated is `path/to/file.s`, the compiler should place this file at `o/u/t/path/to/file/.s`.

- `-O`: Disable optimizations.
- `-target <OS>`: Specify the operating system for which to generate code.

  `OS` may be one of `linux`, `windows`, and `macos`. Your compiler is only required to support the `linux` option. You may support additional operating systems at your discretion, and you may define the default operating system for your compiler in a way that is convenient to you.

11 Build script

Your build script `xic-build` from previous programming assignments should remain available. The expected behaviors of the build script are as defined in the previous assignment. The build script must be in the root directory your submission `zip` file. Problems within the test script from previous submissions should be fixed.

12 Functional test contest

Now that your compilers will generate runnable code, you will be able to include end-to-end tests (often called functional tests) as part of your testing strategies. For fun (and for good karma), groups may participate in a contest of sorts. Each group may submit a subset of their most difficult test cases, and the course staff will run these test cases against every group's compiler. The goal is to expose bugs in other groups' compilers using test cases that conform to the standard specifications.

Each functional test case will be a valid Xi source file. A compiler `c` passes a test `t` if and only if

- `c` successfully compiles `t` into an assembly file `a`,
- assemblling and linking `a` against the standard Xi library results in a runnable program `o`, and
- when executed, `o` terminates with a exit code 0 within an arbitrary bounded amount of time, i.e., it terminates normally, and not as a result of assertion failing or an array out-of-bounds violation.

All test cases must

- be ASCII-encoded files,
- be valid Xi programs, according to the standard specifications (i.e., the Xi language specification and Xi type system specification),
- use only standard `io` and `conv` interfaces,
- not read input,
- contain at most 10 lines of code, excluding comments, and
- contain no line longer than 80 characters.

Each group may submit up to 5 test cases. Submitting test cases that fail to conform to the above requirements may result in disqualification (and thus bad karma). Groups that submit test cases that expose bugs in compilers, i.e., test cases that compilers do not pass, will discover good karma. All test cases submitted for the contest will be released after the assignment's due date.

13 Submission

You should submit these items on CMS:

- `overview.txt/pdf`: Your overview document for the assignment. This file should contain your names, your NetIDs, all known issues you have with your implementation, and the names of anyone you have discussed the homework with. It should also include descriptions of any extensions you implemented.
- A `zip` file containing these items:
  - *Source code*: You should include all source code required to compile and run the project. Please ensure that the directory structure of your source files is maintained within the archive so that your code can be compiled upon extraction. If your code depends on any third-party libraries, please include compilation instructions in your overview document.
    Include your parser generator input file, e.g., `*.cup`, as well as the generated code. If you use a lexer generator, please include the lexer input file, e.g., `*.flex`, as well as the generated code.
  - *Tests*: You should include all your test cases and test code that you used to test your program. Be sure to mention where these files are and to describe your testing strategy in your overview document.
  - *Contest test cases* (optional): Up to 5 functional test cases your group would like to be considered for the contest. These test cases must reside in directory `contest` at the root of the `zip` file directory hierarchy.

  Do not include any non-source files or directories such as `.class`, `.classpath`, `.project`, `.git`, and `.gitignore`.
- `pa5.log`: A dump of your commit log since your last submission from the version control system of your choice.