Using WebAssembly to Teach Code Generation in a Compiler Design Course

Ariel Ortiz
Department of Computing, School of Engineering and Sciences
Tecnologico de Monterrey, Campus Estado de México
Atizapán de Zaragoza, Estado de México, Mexico
ariel.ortiz@tec.mx

ABSTRACT

For many students, code generation is the most demanding topic covered in a typical undergraduate *Compiler Design* course. In this poster, the author will present how he has successfully used the fairly novel and promising WebAssembly technology in his course in order to make the said topic more relevant and engaging for learners. Once we have a compiler frontend that produces an abstract syntax tree (AST) and a symbol table for a given source program, the code generation phase involves traversing the AST and emitting the corresponding WebAssembly instructions in a generally straightforward fashion. The code generation phase is significantly simplified given that WebAssembly is actually an intermediate code for a stack-based virtual machine. The real machine code is produced later by the WebAssembly runtime's just-in-time (JIT) compiler. During execution, the generated code may call functions written in a high-level language directly supported by the runtime. This allows having basic I/O capabilities, like printing to the screen or reading input from the keyboard. At the end of the semester, students have authored the frontend and backend of a fully working compiler that translates a simple C-like procedural language into platform-neutral executable WebAssembly code.

CCS CONCEPTS

• **Software and its engineering → Source code generation**.

KEYWORDS

Compiler Design, Code Generation, WebAssembly

1 INTRODUCTION

WebAssembly [7] (Wasm for short) is a binary code format specification released in 2017. This technology can be implemented in web browsers or standalone applications in a secure, open, portable, and efficient fashion [3]. Wasm was primarily designed as a compilation target, so using it for code generation in a *Compiler Design* course is quite suitable and makes compiler projects more manageable for students.

Code generation is a topic that has been extensively covered in compiler construction books in the past [1, 2], although most of these texts don't emphasize on how to produce code for a new generation of stack-based virtual machine instruction sets such as those for the *Java Virtual Machine* (JVM) [6], the *Common Language Infrastructure* (CLI) [5], the *Yet Another RubyVM* (YARV) [8] or WebAssembly.

ACM ISBN 978-1-4503-9071-2/22/03.
https://doi.org/10.1145/3478432.3499119

2 GENERAL OVERVIEW

In the first part of the course, students write the compiler frontend which creates the AST and a symbol table starting from an input source program. They do this by handcrafting in the C# programming language a *recursive descent parser* [1]. Afterwards, they apply the *visitor design pattern* [4] to write the compiler backend in order to traverse the AST and emit the corresponding *Wasm text format* [7] instructions.

The generated textual code is translated into Wasm binary code and executed using the *Wasmer* Python package [9], a Wasm standalone runtime. Wasmer also allows us to write a simple runtime library using Python in order to provide basic I/O and memory management facilities. Although Wasm only has direct support for integers and floats, provision for strings and arrays can be achieved by using a *resource handle* mechanism that is capable of interacting with the runtime system.

Anecdotal evidence shows that this approach to code generation has been well received by students. At the end, their projects are able to compile to Wasm a variety of interesting programs, including: converting to binary numbers, computing factorials, checking for palindromes, and sorting arrays. More information available at: `arielortiz.info/sigcse2022`

REFERENCES

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools* (2nd. ed.). Addison-Wesley, Boston, MA.
[2] Keith D. Cooper and Linda Torczon. 2011. *Engineering a Compiler, Second Edition.* (2nd. ed.). Morgan Kaufmann, San Francisco, CA.
[3] Colin Eberhardt. 2019. *What Is WebAssembly?* O'Reilly Media, Sebastopol, CA.
[4] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. 1995. *Design Patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley, Boston, MA.
[5] ECMA International. 2012. ECMA-335 Common Language Infrastructure. Retrieved December 9, 2021 from https://www.ecma-international.org/publications-and-standards/standards/ecma-335/
[6] Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley, and Daniel Smith. 2021. The Java Virtual Machine Specification, Java SE 17 Edition. Retrieved December 9, 2021 from https://docs.oracle.com/javase/specs/jvms/se17/html/
[7] Mozilla and individual contributors. 2021. WebAssembly. Retrieved December 9, 2021 from https://developer.mozilla.org/en-US/docs/WebAssembly
[8] Koichi Sasada. 2005. YARV: Yet Another RubyVM: Innovating the Ruby Interpreter. In *Companion to the 20th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05).* ACM Press, New York, NY, 158–159. https://doi.org/10.1145/1094855.1094912
[9] Wasmer, Inc. 2021. Wasmer-Python GitHub Repository. Retrieved December 9, 2021 from https://github.com/wasmerio/wasmer-python