Web Assembly Virtual Machine

Compiler Construction '17 Final Report

Deniz Ulusel   Malte Tobias Brodmann   Sabit Gokberk Karaca
EPFL
{deniz.ulusel, gokberk.karaca, malte.brodmann}@epfl.ch

1. Introduction

The first part of our compiler project is based on 6 different Amy code into WebAssembly code that can be run using NodeJs. We extend our compiler in a way such that the program will be directly run after all previous phases of the compiler successfully ended. The difference from our approach to an interpreter is that we introduce a new phase at the very end of the compilation process that takes a Module generated by the CodeGen phase as an input and executes the included WebAssembly code within a Virtual machine. The Virtual machine we created tries to take as many conceptually ideas from WebAssembly into account as possible i.e. we use data structures and construct such as a stack (used by several instructions) and a data memory as described in the WebAssembly specification. The result of compiling an input program with our extended compiler should be the same as using the interpreter option from the Amy reference compiler.

2. Examples

Using the extended compiler will result in the compilation of the input program and the execution of the generated WebAssembly code. Potential inputs and outputs will be displayed on the console.

```
sbt:amyc> run lib/Std.scala exmpls/test.scala
[info] Running amyc.Main lib/Std.scala exmpls/test.scala
−> What is your name?
<− James
−> Hello James! And how old are you?
<− 25
−> 25 years old then.
```

3. Implementation

3.1 Theoretical Background

A virtual machine is a program that is based on a real computer architecture and emulates a physical computer in software. A process virtual machine is a virtual machine that is designed to execute the same representation of a program across different platforms [Wikipedia contributors 2017].

WebAssembly is a specification for an assembly language and a corresponding machine language that is meant to be run on a virtual machine in web browsers. with the goal of enabling languages such as C and C++ to be run on web browsers, as well as speeding up execution of web applications [Wikipedia contributors 2018].

3.2 Assumptions

For now we have several assumptions regarding the WebAssembly input code of the VM phase of our compiler:

(1) The set of possible WebAssembly instructions is very restricted. We only expect instructions as an input that are also generated by our `CodeGeneration` phase. This is why all instructions interacting with the main stack will only be the corresponding i32 versions.

(2) We assume that `Loop` blocks will never be inside of either branches of an `If Else` instruction pair.

3.3 Implementation Details

3.3.1 VM Architecture

The virtual machine relies on five main data structures to execute given programs. The structures are defined as follows:

Instruction memory
The instruction memory is modeled as an array of `Instruction` objects as defined in the `wasm` package. Its size is determined by the length of the input program after preprocessing. A pointer to the instruction memory, called the program counter, is used to keep track of the next instruction to be executed.

Data memory
The data memory is modeled as an array of bytes

Main stack
The second item

Call stack
The sadasd

Global memory
The third etc ...

3.3.2 Code Execution

Before the code execution starts, all instructions of the input program are loaded into the instruction memory. To do so we place all instructions in each function of the input module back to back in the Instruction Memory. The main function is placed at the beginning of the instruction memory and therefore the program counter is initialized to 0. After this the execution of the program starts by executing the first instruction of the main method and continues until the last instruction of the main method is executed. In order to indicate the end of the functions, a `Return` instruction is added to the end of each function. Each execution of an instruction returns an integer value that is assigned to the program counter in order to choose the next instruction that is going to be executed, i.e. normal instruction will increase the program counter by one whereas instructions as `Call` or `Branch` change the program counter to the instruction they are referring to. Although each instruction is handled differently by the virtual machine, it is possible to separate them into groups depending on their effects on the hardware.

Numeric/Logical Instructions
All of these instructions consume two i32 operands from the main stack and apply the operation on these operands. The instruction that is on top of main stack is considered as the second operand of instruction. The result is pushed to the main stack when the execution is finished. After the execution, program counter is increased by 1 and returned back to indicate the next instruction. These instructions does not change the order of execution.

Control Instructions
These instructions are mainly used in order to change the flow of execution.

`If` instructions are used to decide which branch of the Web Assembly code will be executed and they are the most complex instruction in this group in terms of their implementation. To be able to execute `If` instructions correctly, a helper method is used. This helper method is responsible for iterating over the Web Assembly code of the If-Then-Else construct and finding the corresponding `Else` and `End` instructions. This helps us to determine the beginning and ending indices of this execution block. Then a conditional value is consumed from the top of the main stack. If this value is different from 0 the `Then` branch is executed, otherwise the `Else` branch is executed. All instructions of the other branch are not executed in this case. We skip them using the indices which are indicating the boundaries of both branches. A problematic case is a `Branch` instruction inside of the executed block. This is because the VM will try to execute all instructions inside of one branch. Executing the Branch instruction might change the program counter in a way such that we jump out of the If-Then-Else construct. For that reason we introduced a variable `isBranch` that verifies that we remain inside of the correct Branch.
In order to optimize the process of finding matching `If`, `Else` and `End` instructions, a map is used. If the indices have been already found during a previous execution, they are simply retrieved from the map. If they don't exist in the map, the helper function is called and indices are added to the map.

`Loop` and `Branch` are two additional control instructions. Execution of these instructions is also handled by using a map of labels and their indices. Whenever a `Loop` instruction is encountered, its label is added to the map with its index. Then `Branch` instructions are executed by changing program counter to point the first instruction that comes after the label.

`Call` and `Return` instructions are used in order to start and end execution of a function. Before a `Call` instruction is executed, parameters of callee should be pushed to the main stack. Since this is guaranteed by the Code Generator, it does not require any implementations. After the `Call` instruction is executed, a new frame in the call stack has to be prepared. This frame contains the return address of callee, number of local variables of caller and number of local variables of callee. After adding these data to new frame, local variables of callee can also be popped from main stack and put into the frame. This completes the preparation for a new function call and program counter needs to be changed to the starting index of callee. Starting indices of each function is stored in a map with their names. Program counter is changed by retrieving the index from map. When execution of a function ends, `Return` instruction is executed. This instruction retrieves the return address from function call frame and sets program counter accordingly. Execution of caller continues after this point.

Getter and Setter Instructions

4. Possible Extensions

References

Wikipedia contributors. Virtual Machine, 2017. URL `https://en.wikipedia.org/wiki/Virtual_machine`. [Online; accessed 11-January-2018].

Wikipedia contributors. Web Assembly, 2018. URL `https://en.wikipedia.org/wiki/WebAssembly`. [Online; accessed 11-January-2018].