Final Year Project Report

Full Unit - Interim Report

Scribble Programming Language

Blake Loring

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Adrian Johnstone

Department of Computer Science
Royal Holloway, University of London

February 20, 2014

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 8,674

Student Name: Blake Loring

Date of Submission:

Signature:

Table of Contents

Abstract

The aim of the project is to create a platform independent programming language, parser and virtual machine capable of being embedded as a scripting language inside larger C++ applications.

Chapter 1: **Introduction**

Chapter 2: **Instruction Encoding**

The instruction encoding is one of the most vital parts of a virtual machine.

Every operation that the virtual machine is able to perform from addition to memory allocation is able to be represented as an instruction or a set of instructions within the virtual machine. These instructions are executed one after another by the processor each one performing an action which will change the state of the virtual machine, for example one instruction may jump to an earlier set of instruction while another may load an integer constant like the number 5. This means that the instruction set and instruction encoding for a virtual machine has a very significant impact on the speed and compiled code size of your programs.

The most common approach to instruction encoding is to represent every instruction and the arguments required for the instruction to operate within a fixed size region of memory. Having fixed size instructions can lead to wasted space however it makes instruction stepping and jumping significantly simpler because the virtual machine can calculate the location of the next instruction without needing to inspect the instructions which are being skipped.

One thing that can have a dramatic effect on the instruction encoding is whether the virtual machine is stack or register based. A stack based virtual machine will store all of the data used in operations on a stack for example to perform the expression 5 + 4 you would load 5 to the stack, load 4 to the stack and the perform the add operation which would pop the top two values from the stack and add them together then place the result of the addition back into the stack. This approach leads to extremely small instruction sizes, usually less than 32 bits, as instructions do not require much additional information to execute, this can however lead to a larger number of instructions being used because of the stack operations required to make the program run. A register based virtual machine instead chooses to emulate a real hardware CPU more closely. In this model the virtual machine has a fixed number of registers and instructions modify these instead of the stack for most operations. The example given about the stack would instead be load 5 into register 0, load 4 into register one, add registers 5 and 4 and place the result into register 2. This approach requires larger instructions because they have to carry data about the registers they are operating on however it potentially decreases the number of instructions that have to be generated. It has also been argued that register based instruction sets translate better to machine code when used with a ByteCode to native code compiler used in technologies such as JIT as is common within modern virtual machines. Which model is better is a wide area of debate and there is no clear answer as each has clear advantages and disadvantages and strong support for both sides from developers with languages like Java using stack based virtual machines and technologies like Microsofts .NET and Googles Dalvik using register based virtual machines.

Scribble uses a register based virtual machine, I decided to use a register based virtual machine over a stack based virtual machine as I was more familiar with register based environments.

The instructions in Scribble are 64 bits wide, the first 8 bits always represent the operation to be perform (For example OpAdd, OpNewArray, etc) and the purpose rest of the data will vary depending on the instruction. This much space per instruction is probably not necessary and the wasted space will result in an increased size of compiled programs over equivalent programs in other languages such as LUA however when I was developing the instruction set I did not know what constraints on instructions I would find acceptable and decided to give myself as much overhead as seemed reasonable.

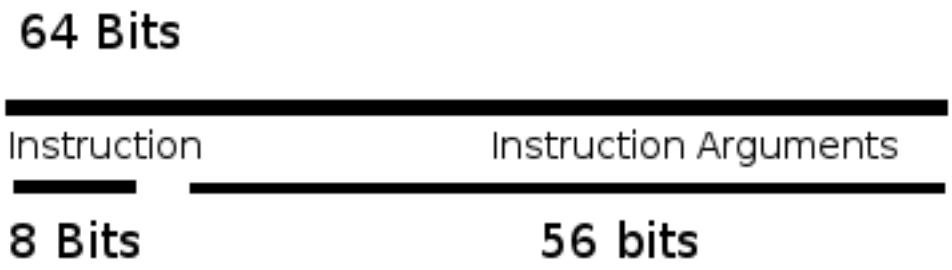

Fig 1. Scribble Instruction

In order to make the virtual machine useful I had to ensure that there where enough instructions available to achieve anything that was valid within the higher level language. This involved looking at each of the language features and working out a way of expressing them as a set of instructions, for example the for loop is achieved by using assign, compare and jump operators in order to iterate until a condition is met. The number and complexity of the instructions that a virtual machine can perform can have a significant impact on its performance. If the instructions are to specialized then a larger number of possible instructions will be needed which could potentially slow down the virtual machine as it has to spend more time with each instruction or increase the size required to store the part of the instruction that tells the virtual machine what to do. Alternatively having less complex instructions will increase the number of instructions that need to be executed to perform each task which and could potentially lead to the virtual machine not being able to complete a task or support a feature later on.

Chapter 3: **Type Inference**

Chapter 4: **Package Handling**

Chapter 5: **Garbage Collection**

Bibliography

[1] Bison - The GNU Parser generator website http://www.gnu.org/software/bison/

[2] Flex - The fast lexical analyser website http://flex.sourceforge.net/

[3] The MSDN resource on data type ranges http://msdn.microsoft.com/en-us/library/s3f49ktz.aspx

[4] The GoLang specification http://golang.org/ref/spec

[5] A No-Frills Introduction to Lua 5.1 VM Instructions http://luaforge.net/docman/83/98/ANoFrillsIntroToLua51VMInstructions.pdf

[6] Java Virtual Machine - Wikipedia Article - http://en.wikipedia.org/wiki/Java_virtual_machine

[7] V8 Javascript Engine - https://code.google.com/p/v8/