Java Direct Bytecode Execution Extension on RISC-V

by

Joshua Daniel Wong

A thesis submitted in partial satisfaction of the

requirements for the degree of

High School Diploma

in

Computer Science

in the

Technology

of

Don Bosco Technical Institute,

Computer Science and Electrical Engineering

Committee in charge:

Brendan Chua, Dean
Marwan Shawky, Chair
Ed Sepulveda, Instructor

Spring 2023

The thesis of Joshua Daniel Wong, titled Java Direct Bytecode Execution Extension on RISC-V, is approved:

Dean _____ Date _____

Chair _____ Date _____

_____ Date _____

Don Bosco Technical Institute,
Computer Science and Electrical Engineering

Java Direct Bytecode Execution Extension on RISC-V

Abstract

Java Direct Bytecode Execution Extension on RISC-V

by

Joshua Daniel Wong

High School Diploma in Computer Science

Don Bosco Technical Institute,
Computer Science and Electrical Engineering

Brendan Chua, Dean

Marwan Shawky, Chair

To Me

Contents

Contents **ii**

List of Figures **iii**

List of Tables **iv**

1 Introduction **1**
1.1 Reasons for Development 1
1.2 Practical Uses 1
1.3 Objectives and Future Research 2

2 Java **4**
2.1 Java Virtual Machine 4
2.2 Conclusion 4

List of Figures

List of Tables

Acknowledgments

Chapter 1

Introduction

This thesis proposes the addition of direct Java bytecode execution to the RISC-V architecture. The extension, named "JC" (Java on Core), builds on top of the future "J" extension, which is designed to facilitate the development of dynamically translated languages. By incorporating JC, developers will have a compelling reason to utilize Java, as Java's runtime performance will be significantly improved.

1.1 Reasons for Development

The development of the JC extension stems from several key motivations. First of all, Java is a widely adopted programming language, particularly in enterprise-level software development. By enabling direct Java bytecode execution on the RISC-V architecture, the platform becomes more appealing to Java developers, who can leverage the enhanced runtime performance and compatibility.

Secondly, the RISC-V architecture aims to be versatile and support a wide range of applications. While the initial focus has been on embedded systems for specific tasks, the addition of the JC extension expands the architecture's capabilities to include high-performance Java execution.

Finally, the incorporation of the JC extension aligns with the broader goal of promoting the RISC-V architecture as a comprehensive and flexible platform that can accommodate diverse programming languages and execution environments.

1.2 Practical Uses

The introduction of direct Java bytecode execution via the JC extension opens up various practical applications. Some of the potential use cases include:

1. Increased Android ART Performance: Java applets when translated to run on RISC-V with the "JC" extension will have greatly accelerated performance. Allowing the use

of Java bytecode directly on RISC-V will also improve performance during translation. By using the JC extension, ART can skip some pre-implemented JC instructions on RISC-V, allowing for decreased startup and update times.

2. Improved Java Performance: Java applications running on RISC-V with the JC extension can benefit from accelerated runtime performance. By leveraging the architecture's features and optimizations specific to Java bytecode execution, developers can achieve enhanced efficiency and responsiveness in their Java applications.

3. Enterprise-level Software: The RISC-V architecture with the JC extension becomes an attractive choice for enterprise-level software development, where Java is frequently used. The improved performance and compatibility can lead to more efficient execution of large-scale applications, resulting in better productivity and cost-effectiveness.

4. Integration with Existing Java Ecosystem: Java has a rich ecosystem of libraries, frameworks, and tools. With the JC extension, developers can seamlessly integrate their existing Java codebase and leverage the vast array of Java resources available, enabling faster development and reducing the effort required for migration to the RISC-V architecture.

1.3 Objectives and Future Research

The primary objective of this thesis is to design and implement the JC extension, enabling direct Java bytecode execution on the RISC-V architecture. Key research areas to be explored include:

1. Instruction Set Design: The design of the JC extension requires careful consideration of the instructions needed to support Java bytecode execution efficiently. This research will involve analyzing the requirements of the Java Virtual Machine (JVM) and writing them to the RISC-V architecture.

2. Runtime Optimization: Optimizations specific to Java bytecode execution will be investigated to improve the performance of Java applications running on RISC-V. This research will focus on techniques such as just-in-time (JIT) compilation, adaptive optimization, and garbage collection strategies tailored for RISC-V.

3. Compatibility and Interoperability: Ensuring compatibility with existing Java bytecode and the Java API is essential. This research will explore techniques to achieve seamless integration with the existing Java development tools, libraries, and frameworks, allowing developers to leverage their prior knowledge and resources.

4. Evaluation and Benchmarking: The performance of the JC extension will be evaluated through benchmarking against existing JVM implementations. This research will in-

volve designing comprehensive benchmarks to assess the runtime performance, memory utilization, and overall efficiency of Java applications on RISC-V with the JC extension.

By addressing these objectives and conducting further research in these areas, this thesis aims to lay the foundation for incorporating direct Java bytecode execution into the RISC-V architecture, thereby enhancing its appeal to Java developers and expanding its range of practical applications.

Chapter 2

Java

2.1 Java Virtual Machine

2.2 Conclusion