Angebot
Dynamic Binary Translator: RISC–V → x86

10. Juni 2020

Bachelor-Praktikum Rechnerarchitektur (IN0005)
Großpraktikum

Noah Dormann, Simon Kammermeier,
Johannes Pfannschmidt, Florian Schmidt

Projektspezifikation

Im Rahmen dieses Projektes wird ein Übersetzungssystem entwickelt, welches für die RISC-V-ISA kompilierten Code innerhalb einer x86-64-Umgebung lauffähig macht.

Zu diesem Zwecke soll dynamische Binärübersetzung eingesetzt werden. Die ausführbaren und tatsächlich angesprungenen Programmteile der ELF-Datei sollen also entsprechend eines Mappings einzelner Instruktionen dynamisch in x86-64-Code übersetzt werden. Diese Programmteile werden im Rahmen der Übersetzung zwischengespeichert, was somit sicherstellt, dass ein Ausführungsblock nicht doppelt übersetzt werden muss. Zudem sollen diverse Optimierungsansätze zur Verbesserung der Übersetzungs- und Ausführungsgeschwindigkeit herangezogen werden.

Minimale Zielsetzung

Ausführung einfacher RISC-V-Programme Wir möchten die Unterstützung von einfachen, für die RISC-V-ISA kompilierten ELF-Dateien ermöglichen. Dies setzt voraus, dass wir in der Lage sind, die grundlegenden Instruktionen des RV32I/RV64I zu übersetzen. Um die ganzzahlige arithmetische Funktionalität der ISA abzurunden, möchten wir auch die M-Erweiterung des Instruktionssatzes (Multiplikations-, wie Divisionsbefehle) unterstützen. Verwendet die ELF-Datei nicht unterstütze Instruktionen, möchten wir entsprechende Ausgaben liefern, die gegebenenfalls auf eine mögliche Kompilierung des Codes unter abgeschalteten Befehlssatzerweiterungen hinweist.

Unterstützung von system calls Im Rahmen der minimalen Zielsetzung möchten wir environment calls via der `ECALL`-Instruktion zu den Linux system calls `write`, `read`, `printf`, `open`, `close` sowie `fstat` anbieten. Dies steht in direkter Verbindung mit der obigen Ausführung einfacher Programme: es gibt schließlich kein „Hello, World!“ ohne `write`.

Benchmark via gzip Zum Zwecke der Performancemessung des entwickelten Übersetzers sowie zur Quantifizierung potenziell folgender Optimierungen möchten wir die RISC-V-64-Version von gzip als Benchmark heranziehen.

Dokumentation anfertigen.

Ausnahmen und Einschränkungen

Entgegen des obigen ersten Punktes möchten wir die mit den `CSR`-Registern verbundene Funktionalität nicht unterstützen. Nicht unterstützte Befehlssatzerweiterungen werden vor dem Programmstart abgefangen und an den Benutzer gemeldet.

System calls, die nicht in der obigen Aufzählung enthalten sind, möchten wir vorerst wie `NOP`s behandeln[1].

Erweiterte Zielsetzung

Unit-Testing Um die Korrektheit unseres Systems zu verifizieren, wie auch die interne Fehlersuche zu erleichtern, möchten wir Unit-Tests für unsere verschiedenen Systemkomponenten schreiben. Hierbei geht es vor allem um den Code-Cache, den Parser der RISC-V-Instruktionen sowie die dynamische Codegenerierung, nachdem die einwandfreie Funktionalität hier von zentraler Wichtigkeit ist.

Floating Point Extension Um die arithmetische Funktionalität zu vervollständigen, möchten wir nach Möglichkeit die „F“ bzw. „D“-Standard Extensions zur Gleitkommaarithmetik unterstützen.

[1] Evidenterweise soll die encodierte Version `NOP`-Pseudoinstruktion `ADDI x0, x0, 0` nicht übersetzt werden.

Weiteres Benchmarking Die Unterstützung der Fließkommaarithmetik öffnet die Tore zu komplexeren Benchmarkingoptionen – insbesondere ist hier die Möglichkeit von Image Processing Benchmarks gegeben.

Diverse Optimierungen Im Laufe des Projekts möchten wir alle Programmteile auf Performanz optimieren. Insbesondere die Übersetzungsstrategien sollen hierbei im Mittelpunkt stehen. Als Ansatzpunkte seien an dieser Stelle Konzepte wie Chaining, return address prediction, wie auch das Übersetzen ganzer Schleifen (statt nur einzelner Basisblöcke, die nach jeder Iteration erneut angesprungen werden müssten).

Zeitplan

April 2020	Einlesen in die System-V-ABI[2] sowie die unpriviligierte RISC-V-Spezifikation[3]; Aufsetzen der Projektstruktur und Einarbeitung in die Systematik
Mai 2020	Implementierung des Caches, Parsers, ELF-Loaders, Blockloaders sowie der Übersetzungen für die Instruktionen der RV64I
Juni 2020	Integration der Komponenten, Umsetzung der system calls, erstes Benchmarking
Juli 2020	Abschluss des obigen Minimalanteils (RV64I, system calls, gzip), Beginn der Dokumentation
August 2020	Anfertigung der Dokumentation sowie Behandlung der erweiterten Funktionalität

[2] `https://github.com/hjl-tools/x86-psABI/wiki/x86-64-psABI-1.0.pdf`, Stand 09.06.2020.
[3] `https://content.riscv.org/wp-content/uploads/2016/06/riscv-spec-v2.1.pdf`, Stand 09.06.2020.