Semantics of an embedded vector architecture for formal verification of software

A Part III project proposal

G. M. Brown (*gmb60*), Queens' College

Project Supervisor: Dr J. D. Yallop

Abstract

All good implementations of cryptographic algorithms must be correct, side-channel free and fast. Most cryptographic libraries focus on maximising speed by writing hand-tuned assembly. This can introduce subtle bugs that invalidate correctness or introduce side-channels. Whilst tools exist to help formally verify these algorithms, none are designed to target the recent M-profile Vector Extension for the Armv8.1-M architecture. My project seeks to define semantics for these vector instructions, designed to be used for formal verification of software. I will use these semantics to formally verify the correctness of hand-written assembly for cryptographic applications.

1 Introduction, approach and outcomes

In almost all cases, the best implementation of an algorithm will be both correct and fast. If an implementation is not correct, then it implements a different algorithm. If an implementation is slow, then it wastes resources that could be used to perform additional work. Ensuring an implementation satisfies both properties is usually difficult.

This is especially true when it comes to writing assembly code, to maximise performance. Truly optimal performance comes when details of the processor microarchitecture are used to reorder instructions to minimise the amount time the processor is stalled. After shuffling the instructions, it can be incredibly difficult to recognise the original algorithm and the intent of each instruction. This leads to subtle bugs which can invalidate the correctness and other safety properties of the implementation.

To eradicate the correctness bugs with certainty, formal verification is necessary. Formal verification starts with a description of the semantics of machine code — a model of the action of instructions on machine state. This semantic framework is incorporated into a larger logical system, such as a higher-order-logic theorem prover like Isabelle or a dependently-typed programming language like Agda. These logic systems can then be used to formally prove that given assembly code satisfies the desired correctness properties, along with any other safety property. For instance, formal verification is often used in cryptography to prove the absence of side-channels.

Formal verification tools already exist for some architectures. For example, Jasmin is a tool for x86 that accepts assembly language augmented with logical assertions and high-level constructs like variables and loops. This is converted into pure assembly language output, as well as a proof that all the assertions in the input hold.

An often-overlooked architecture in the field of formal verification is Armv8.1-M. This instruction set is designed for use by microcontrollers, which operate in a resource-constrained environment.

In particular the recent M-profile Vector Extension (MVE), which provides SIMD instructions for Armv8.1-M cores, has no known semantics suitable for the formal verification of software.

My project is to resolve this issue by using Agda to develop a semantics for MVE instructions suitable for formal verification. This entails the design of operational and axiomatic semantics, and proof of an equivalence between these two approaches. Operational semantics describe the explicit effect of instructions on the state of the machine, whereas axiomatic semantics describe how execution effects properties of the state.

I will also use the two semantic systems to formally verify the correctness of several algorithms used in cryptography, with implementations produced by Becker et al. [1]. These include Barrett reduction, Montgomery multiplication and the number-theoretic transform.

2 Workplan

Start	End	Work
2021-12-06	2021-12-19	Familiarise myself with MVE instructions. Begin work on defining the operational semantics, focusing on representing machine state and auxiliary definitions.
2021-12-20	2022-01-02	Break for Christmas and New Year's.
2022-01-03	2022-01-16	Finish the definition of the operational semantics. Prove some useful lemmas regarding manipulating the structure of blocks of assembly.
2022-01-17	2022-01-30	Define the axiomatic semantics in the form of Hoare triples.
2022-01-31	2022-02-13	Prove and equivalence between the operational and axiomatic semantics.
2022-02-14	2022-02-27	Progress review. Begin formal verification of Barrett reduction.
2022-03-14	2022-03-27	Formally verify an implementation of Montgomery multiplication.
2022-03-14	2022-03-27	Begin work to formally verify one iteration of the number-theoretic transform, which is a form of discrete Fourier transform in the domain of integers modulo a prime.
2022-03-28	2022-04-10	Finish verification of one iteration of the number-theoretic transform.
2022-04-11	2022-04-24	Write-up.
2022-04-25	2022-05-08	Write-up.
2022-05-09	2022-05-22	Contingency weeks.
2022-05-23	2022-05-27	Submission. Begin work on presentation.

References

[1] Hanno Becker et al. *Polynomial multiplication on embedded vector architectures*. Cryptology ePrint Archive, Report 2021/998. 2021. URL: https://ia.cr/2021/998.