Project Documentation
Decoder Implementation of the RISC-V Efficient Trace

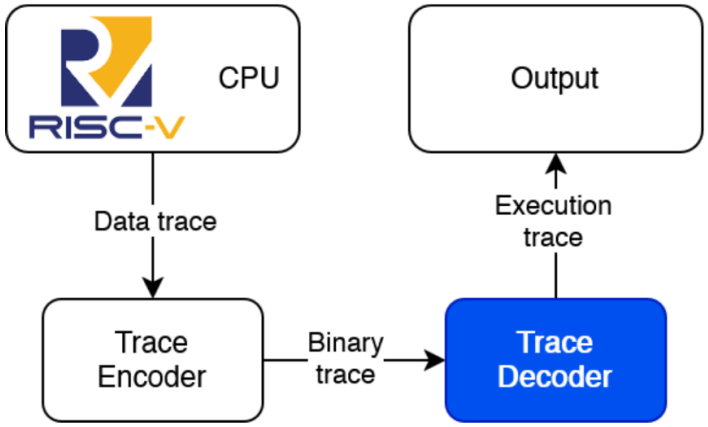
Righi Samuele / UNIBO, Dipartimento di Informatica - Scienza e Ingegneria

July 1, 2025

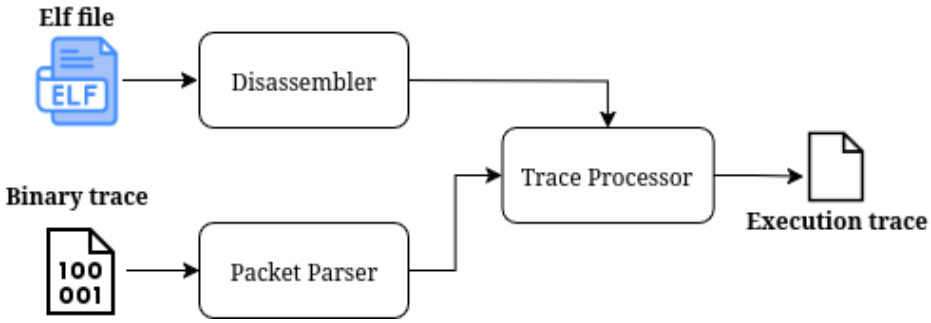
Contents

1 System Overview

The goal of this project [1] is to implement the *Efficient Trace for RISC-V* [2] specification.



More specifically, it focuses on developing a trace decoder that converts the binary trace data generated and published by the encoder into structured packets. These packets are then used to reconstruct the program's execution trace at the assembly level.



Main components of the system include:

- **Packet Parser**: Parses the binary trace output generated by the trace encoder and reconstructs the corresponding trace packets. It acts as a deserializer, converting the raw binary stream into structured packets according to the formats and subformats defined in the Efficient Trace for RISC-V specification [2, Chapter 7].
- **ELF Disassembler**: Disassembles the input ELF binary and extracts the relevant sections, such as instruction addresses and symbol information. It uses `pyelftools` to parse ELF metadata and `Capstone` to disassemble the binary code, generating a mapping of the form {`address: (opcode, attributes)`}.
- **Trace Processor**: Combines the decoded trace packets with the disassembled ELF information to reconstruct the instruction-level execution trace. This component essentially implements the reference pseudocode defined in the Efficient Trace for RISC-V specification [2, Chapter 11].

2 Repository Structure

The repository is organized into multiple directories and files, each serving a specific purpose in the trace decoding pipeline:

- `main.py`: Entry point of the application. It orchestrates the decoding process using the components defined in the `src` directory.
- `disassembler_config.yaml`: Configuration file for the disassembler, specifying sections to extract and parameters for parsing the ELF file.
- `requirements`: Lists the Python dependencies required to run the project.
- **src/**: Contains the core source code, organized into the following modules:
  - **controller/**: Contains the `trace_decoder.py`, which serves as the control layer and entry point for initializing and running the decoding process.
  - **domain/**: Defines data models, constants, enumerations, and packet structures used throughout the system. Notable files include:
    - `packet_format.py`: Defines the format and parsing logic of trace packets.
    - `trace_processor_model.py`: Contains data structures for representing decoded trace states.
    - `const.py`, `enums.py`: Define shared constants and enumerated types.
  - **services/**: Implements the functional logic for each major component:
    - `packet_parser.py`: Parses the raw binary trace and constructs structured packets.
    - `elf_disassembler.py`: Disassembles ELF files and extracts the required sections.
    - `trace_processor.py`: Reconstructs the instruction-level trace by matching packets to disassembled instructions.
    - `trace_processor_utils.py`: Helper utilities for trace processing.
    - `instruction_logger.py`: Logs the reconstructed instruction stream.
- **tests/**: Contains test cases and example workloads to validate the decoder. Each subdirectory includes:
  - An ELF binary file (`.riscv`)
  - The corresponding trace dump (`.dump`)
  - The raw binary trace (`packets.bin`)

  Subfolders include:
  - `gpios_all/`
  - `hello_culsans/`
  - `l1_test/`

3 Project Setup

To set up and run the RISC-V Efficient Trace Decoder, follow the steps below. This section assumes a working Python 3 environment and access to the repository. It's `raccomended the use of a virtual environment` to avoid polluting the system.

3.1 Installation

1. Clone or download the repository, then move into its root directory:

   ```
   cd decoder
   ```

2. Install the required Python dependencies using `pip`:

   ```
   pip install -r requirements
   ```

3.2 Configuration

Before running the decoder, make sure the `disassembler_config.yaml` file is correctly configured. This YAML file specifies which ELF sections to extract and disassemble. A typical configuration may look like:

```
sections:
  - .text
  - .text.startup
```

Ensure that the sections listed correspond to the code sections used to create the binary trace file.

3.3 Running the Decoder

To run the trace decoder on a given test case, execute:

```
python3 main.py <path/to/packets.bin> <path/to/binary.elf>
```

For example:

```
python3 main.py ./tests/hello_culsans/packets.bin ./tests/hello_culsans/
hello_culsans.riscv
```

The binary ELF file and the corresponding binary trace must reside in the same test folder.

3.4 Output

The reconstructed execution trace will be saved in the file:

```
execution_trace
```

This file contains the list of decoded and matched instructions, ordered as executed by the RISC-V processor.

4 Developer Notes

4.1 Unimplemented Features

The following features are currently not implemented:

- Handling of packets format 3, subformat 2
- `report_trap` in the trace processor
- `report_epc` in the trace processor

4.2 Known Issue

The following considerations refer to the encoder version as of July 1, 2025, specifically regarding a known issue with the `gpios_all` test.
This test is expected to fail due to an error in the encoder while generating the `packets.bin` file. The decoder crashes after processing a sync packet (format 3, subformat 0).
Our current hypothesis is that the encoder does not generate the required branch map related to the uninferable discontinuities between the last qualified instruction and the sync packet. According to the specification ([2, Section 9.2]), the encoder should always generate a branch map before a sync packet if there are any unreported branches; however, this does not appear to be the case in this version.

References

[1] R. Samuele, "Decoder implementation of the risc-v efficient trace," July 2025, https://github.com/pulp-platform/rv_tracer/tree/main/decoder.

[2] R.-V. International, "Efficient trace for risc-v," Version 2.0.3, April 19, 2024, https://github.com/riscv-non-isa/riscv-trace-spec.