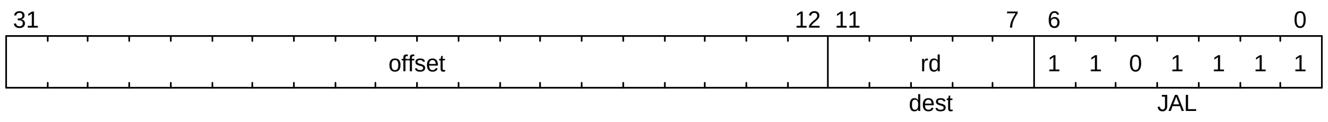
Sail to Asciidoc

Introduction

This document describes how to incorporate Sail source code into ISA manuals written in asciidoc, using the asciidoctor tool. This is primarily aimed at producing documentation for the sail-riscv specification, but can in principle be used for any ISA description written in Sail.

There are two main components to this. First, there is an extension to Sail which produces *documentation bundles* — files indexing the contents of a Sail source file. These documentation bundles are used by an asciidoctor plugin, allowing Sail source code to be included into asciidoc manuals without duplicating the code, which would invariably lead to it getting out of sync over time.

This also enables some fancier features, for example, we can use the information from the Sail source to automatically generate encoding diagrams using asciidoctor diagram. For example:



can be generated from the following Sail:

```
$[wavedrom _ dest JAL]
mapping clause encdec = RISCV_JAL(offset @ 0b0, rd)
  <-> offset : bits(20) @ rd @ 0b1101111
```

which uses a Sail attribute to inform the documentation generator about the labels for the encoding diagram.

Documentation bundles

Documentation bundles are produced by Sail using the `-doc` flag. This works like other Sail target flags, such as `-c` to generate C code, `-coq` for Coq definitions, and so on. For example, if we have three Sail files `a.sail`, `b.sail`, and `c.sail`, we could produce documentation for them using:

```
sail -doc a.sail b.sail c.sail
```

to produce only documentation for `c.sail` (which may still depend on the other files) to type-check, we can use:

```
sail -doc a.sail b.sail c.sail -doc_file c.sail
```

The documentation bundle is a JSON file and will be placed in `sail_doc/doc.json`. The output directory can be changed using the `-o` option, so:

```
sail -doc a.sail b.sail c.sail -doc_file c.sail -o my_doc
```

will produce a file `my_doc/doc.json`. See Sail command line flags for other flags.

Potential workflows

The tooling has been carefully designed to not impose any particular workflow. You can:

- Generate the documentation bundle in advance and check it in. This means documentation contributors will not need Sail to build the documentation. This may lead the bundle becoming out of date, but there are various options here also:
  - You can check if the bundle is up-to-date each time you build the document
  - You can check if the bundle is up-to-date using some kind of continuous integration system
- Have a Makefile (or other build tool) generate the bundle every time the documentation is built.

These are only two possibilities. There are likely many others.

JSON format

The first few keys in the JSON file give information about the documented Sail files, and the state of the repository they are in. These keys can be accessed by custom build tooling to check if the bundle is up to date. As an example:

```
{
  "version": 1,
  "git": {
    "commit": "773a19d17432a1c60cc95a87a587c8255bef9e75",
    "dirty": true
  },
  "embedding": "plain",
  "hashes": { "doc.sail": { "md5": "45ee16b971a5fc084bea556d83f27aff" } },
```

- The `version` key exists so the bundle format can be updated in the future. It is currently always set to 1.
- The `git` key contains the commit hash of the repository within which the `sail` command that produced the documentation bundle was produced. It also contains a flag that is true if the working tree has uncommitted changes. See https://mirrors.edge.kernel.org/pub/software/scm/git/docs/gitglossary.html#def_dirty for details.
- The `embedding` field tells us how other subsequent fields are encoded. See the `-doc_embed` option below for details.
- The hashes field contains a checksum for each Sail file included in the documentation bundle. Like the git commit hash it can be used by tools to check whether the bundle is up-to-date.

The rest of the file contains information for all the documented Sail definitions.

Sail command line flags

- `-doc` — Tells Sail to generate documentation.
- `-doc_file <file>` — Include Sail definitions in `file` in the generated documentation bundle. This option can be passed multiple times. If `-doc_file` is not passed then all files provided to Sail will have documentation generated for them.
- `-doc_embed <plain|base64>` — This option embeds the source code directly within the documentation bundle rather than referencing it from an external file. If `-doc_embed plain` is used then the source and comments is included as-is (with appropriate escaping to be included in a JSON file). With `-doc_embed base64` the source and comments are stored in the JSON as base64 encoded strings. `-doc_embed` is useful if you documentation is separate from the Sail source you are documenting.
- `-doc_compact` — By default the JSON output is pretty-printed with indentation. When this option is used the JSON documentation bundle is printed in a compact form, omitting all unecessary spaces.
- `-doc_format <format>` — This option controls the format for the output. Currently supported options are `adoc` and `asciidoc` (the default) which are the same and both output suitable for the Sail to Asciidoc plugin. Eventually `latex` will also be allowed once the older Sail to Latex documentation generation has been ported to the new documentation system written for this plugin.

Asciidoc commands

The Sail to Asciidoc repository contains several examples demonstrating the various commands in the `examples` subdirectory, with the README in that directory explaining the set of examples.

Syntax highlighting

The Sail to Asciidoc plugin provides a lexer for rouge, an extensible Ruby syntax highlighter supported by asciidoctor and asciidoctor-pdf. If another highlighter with a fixed set of supported languages is used, everything will still work, but without highlighting.