



Authoring and Editing RISC-V Specifications

Version 0., 12/2021: Pre-release version

Table of Contents

Preamble	1
Preface	2
1. Asciidoc authoring for RISC-V contributors	3
1.1. About Asciidoctor	3
1.2. RISC-V AsciiDoc authoring assets	3
1.3. Simply writing	3
1.3.1. Text editors that support AsciiDoc authoring	4
1.4. An asciidoc primer and more	4
1.4.1. Build often	4
2. A few AsciiDoc basics	6
2.1. Paragraphs	6
2.1.1. Basics of blocks and indents	6
2.2. Headers	7
2.3. Hyperlinks and cross references	8
2.3.1. Hyperlinks	8
2.3.2. Cross references	8
2.4. Stem content	9
3. Tables, symbols, and graphics	10
3.1. More on examples	10
3.2. Unicode symbols	11
3.3. Graphics	13
3.3.1. Automated diagramming	13
3.3.2. Wavedrom diagrams in specifications	13
Explanation	15
3.3.3. Graphviz	16
3.3.4. Additional diagram type examples	18
A mention of Bytefeild	20
3.4. Mathematical notations	20
3.4.1. Superscritps and subscripts	20
3.4.2. Latexmath	21
4. Blocks, notes and markers	23
4.1. Blocks	23
4.1.1. Sidebars	23
4.1.2. Admonition blocks	24
4.1.3. Code blocks	27
4.2. Change bars	28
4.2.1. Indicate changes	28
4.2.2. Check for changed lines before a <code>git commit</code>	29
4.3. Footnotes	29
4.4. Index markers	30

4.5. Bibliography and references	31
4.5.1. Automated bibliography procedures with asciidoctor-bibtex	31
4.5.2. Manual bibliography procedures	33
5. Editing Wavedrom diagrams for Unpriv	35
5.1. Relevant contextual information	35
5.2. Example Wavedrom code, before and after	35
5.3. Caveats for editing wavedrom diagrams	37
6. Write simply	39
6.1. Writing is rewriting	39
6.2. Develop style guidelines only as needed	39
6.3. Active voice	40
6.4. Identify information types	40
6.5. RISC-V-specific style guidelines	40
6.5.1. Table captions: above or below the table?	40
Index	42
Bibliography	43

Preamble

This is the preamble, which can contain contributor information and should contain the Creative Commons Attribution statement below.

For this book example, I have copied information directly from several Web sites that are hosted by asciidoctor.org and devoted to providing AsciiDoc/Asciidoctor documentation, and have authored some explanations. Graphics used are either explicitly available for free, are property of RISC-V International, or were created using Wavedrom.

This document is released under a Creative Commons Attribution 4.0 International License.

Preface

This document demonstrates the use of AsciiDoc for RISC-V specifications, with the goal of capturing information that will result in effective and efficient collaboration throughout the community.

Asciidoc is currently the most feature-rich of the popular lightweight markup languages based on markdown. An Open Source effort, it is gaining wider adoption and there is an asciidoc working group. RISC-V also supports some documentation associated with testing that is written in another popular lightweight markup language, Restructured Text.

It's helpful to think of AsciiDoc as [Markdown grown up](#). People in tech often have impulses to re-invent markdown with a brand new lightweight markup language of their own. As appealing as that idea can be, it is inherently flawed. Publishing, like music, can have simple forms, but when fully featured is quite complex. Everyone who has attempted to build upon Markdown to create a simple and feature-rich publishing solution faces the same reality—as we add features, the process necessarily becomes complex.

RISC-V specifications require the use of AsciiDoc markup and the Asciidoctor toolchain with advanced publishing features that are provided by several add-ons. The templates in this repo allow you to jump in with a hands-on approach and build a PDF using the example files.



Because AsciiDoc is gaining in popularity, there are opportunities contributors to the AsciiDoc specification while it is still being developed. You might want to view what Dan Allen and Sarah White are doing. Along with a growing open source community, they support both AsciiDoc and the Asciidoctor toolchain. Feel free to find out about the working group, the specification under development, the toolchain and its various plugins, and other projects that make use of AsciiDoc.

If you'd like to add to the documentation build toolchain, please join the RISC-V group devoted to supporting the RISC-V publishing templates. As members of an Open Source community, we like to support other open source efforts, and for that reason we encourage coordination in adding features so that everyone benefits.



This section is technically called a colophon and is useful as a preface. Its contents should be replaced with your preface contents.

Chapter 1. Asciidoc authoring for RISC-V contributors

As a contributor to RISC-V, your contributions will likely be in one of the two following categories:

- adding content to an existing specification.
- creating an entirely new specification.

In either case, please feel free to examine the AsciiDoc source files for this [example.pdf](#) and adapt the markup for your use.

1.1. About Asciidoctor

AsiiDoc is the markup language and Asciidoctor is a set of toolchains that support publishing from AsciiDoc.

There are three major Asciidoctor toolchains:

- Ruby, which is well-established and which we are using.
- Antora, which is relatively new and which uses NPM.
- Python-based, which is legacy and in maintenance mode.

1.2. RISC-V AsciiDoc authoring assets

Please view the readme in the [docs-templates repo](#) for information on automated build processes.

The above mentioned repo also contains assets (fonts, styles, directory structure, and themes) that you need for RISC-V specifications, along with a [document](#) that takes you through a local install process that supports all of the needed features for a local build.

Although testing your markup by building a pdf is good practice, you can catch many common errors by simply downloading Asciidoctor and building the HTML5 output using the following command:

```
asciidoctor book_header.adoc
```

1.3. Simply writing

To begin authoring in AsciiDoc, all you need is a text editor. [Section 1.3.1, “Text editors that support AsciiDoc authoring”](#) lists editors that support authoring in AsciiDoc.

A quick reference for AsciiDoc markup: docs.asciidoctor.org/asciidoc/latest/syntax-quick-reference/. Most of the markup that you will need is simple, and will have a familiar feel to people who have used git-flavored Markdown. For RISC-V specifications, it is the procedures for [Section 3.3, “Graphics”](#) that add complexity.

The [AsciiDoc Writers Guide](#) contains details about AsciiDoc markup. For pdf generation, you can find details at [asciidocor-pdf](#).

1.3.1. Text editors that support AsciiDoc authoring

I have not yet encountered a true wysiwyg editor for asciidoc. However, there are live preview options listed in the [AsciiDoctor documentation](#). You might want to use your favorite text editor, or perhaps switch to one that has good asciidoc linting.

The following list contains links to resources for text editor/IDE support of asciidoc:

- Information on helpful [AsciiDoc tools that integrate with several popular IDEs](#)
- [VS](#)
- [vim](#)
- [emacs](#)
- [Sublime Text](#) (works best when you add the Sublime Linter package)
- [Atom](#)
- [Notepad++ and gedit](#)
- [OpenOffice](#)--it's possible to start by authoring in OpenOffice and then output to asciidoc, and you can [go the other way](#)
- [AsciiDocFX](#)

We received a request for information on ghostwriter and found that while it does support Markdown, it does not at this time support AsciiDoc.

1.4. An asciidoc primer and more

Much of what you need to know about the asciidoc toolchain that we are using, including install procedures, is online in the

[AsciiDOc Writers' Guide](#)

Here are a few additional, useful links:

- [asciidoc best practices](#)
- [tables](#)
- [links](#)



The best linter for asciidoc is available for Sublime Text, and linters are available for other popular text editors as listed above.

1.4.1. Build often

As soon as you have installed asciidoctor on your computer, you have the ability to check that each individual file builds in html by simply running `asciidoctor filename.adoc` on any file. You can also check that the book or report on which you are working builds in html by running `asciidoctor`

`bookname.adoc` or `asciidoc reportname.adoc` on the book or report header. The strings `filename`, `bookname`, and `reportname` all should be replaced with the actual names you are using.

Asciidoctor has fairly good error messages and usually lets you know the file and row number where the build first breaks.

Chapter 2. A few AsciiDoc basics

AsciiDoc is fully documented, and its documentation is actively maintained. This document contains some information on asciidoc markup to get you started.

For details and additional options, see:

- AsciiDoc/asciidoc [writers' guide](#).
- AsciiDoc [quick reference](#).
- AsciiDoc [user manual](#).

In addition, you have the option of asking questions in the asciidoc discussion list:

As is true of any complex process, asciidoc/asciidoc has some quirks. Please be certain to make use of these templates because they provide you with files that will result in fully featured pdf output.

Best practice is to test the pdf build frequently to ensure that you have not accidentally introduced something that breaks the build.



PDF generation currently requires the use of Ruby 2.7.2.



Please feel free to send questions to help@riscv.org

2.1. Paragraphs

In AsciiDoc, normal paragraphs require no markup at all.

2.1.1. Basics of blocks and indents

If you simply add an indent, your indented text becomes a block like this.

- You can add indented, explanatory paragraphs to lists.

Add a + directly above the line of text that you want to be indented within the list.

- Another point.

Using the + to create an indented paragraph only works within the context of a numbered or bulleted list.

AsciiDoc/asciidoc supports code blocks with syntax highlighting for many languages. You can use either periods or dashes to indicate code blocks, and use macros to indicate that the block contains code in the specified language, as in the following example:

```
[source,python]
....
mono-spaced code block
add a1,a2,a3; # do an ADD
....
```

renders as follows:

```
mono-spaced code block
add a1,a2,a3; # do an ADD
```

See [Chapter 4, Blocks, notes and markers](#) for additional information on blocks.

2.2. Headers

While authoring in asciidoc, you cannot jump directly from a Head 1 to a Head 3 or 4. Your headers must appear in sequence from Head 1 to Head 2, and onward. If you skip over a header in the sequence, asciidoctor throws an error.

Following is an example of a valid sequence of headers.

```
= Title head (book or report title)

[colophon]
= Colophon head (in frontmatter, used for preface)

[[chapter_title]]
== Head 1 (chapter)

=== Head 2 (section)

==== Head 3 (subsection)

===== Head 4 (sub-subsection)

[appendix]
== Appendix title

[index]
Index
```



*Settings in the header file (**book_header.adoc** in the docs-templates repo) trigger auto-generation of Appendix prefixes and of the Index (among other things).*

2.3. Hyperlinks and cross references

Asciidoctor automates some linking as follows:

- Asciidoctor recognizes hyperlinks to Web pages and shortens them for readability.
- Asciidoctor automatically creates an anchor for every section and discrete heading.

2.3.1. Hyperlinks

To create highlighted links, use the pattern in the following example:

```
https://asciidoctor.org[Asciidoctor]
```

You can set [attributes for your external links](#)

2.3.2. Cross references

Use macros for cross references (links within a document) as in the following example:

```
<<Index markers>> describes how index markers work.
```

This renders as:

[Section 4.4, “Index markers”](#) describes how index markers work.

The `book_header.adoc` file in the docs-templates repo sets the `full` cross reference attribute to enable the display of captions from targets in the anchors. This allows you to set captions for tables, blocks, and illustrations. If no caption is provided, Asciidoctor defaults to the *basic* cross reference style.

To set a caption for a table or image, use the pattern as follows:

```
The table below, <<trapcharacteristics,Characteristics of traps>> shows the
characteristics of each
kind of trap.
```

```
[[trapcharacteristics,Characteristics of traps]]
.Characteristics of traps.
[cols="<,^,^,^,^",options="header",]
|===
| |Contained |Requested |Invisible |Fatal
|Execution terminates |No |Nolatemath:[ $\sim{1}$ ] |No |Yes
|Software is oblivious |No |No |Yes |Yeslatemath:[ $\sim{2}$ ]
|Handled by environment |No |Yes |Yes |Yes
|===
```

The table below, [Table 1, “Characteristics of traps.”](#) shows the characteristics of each kind of trap.

Table 1. Characteristics of traps.

	Contained	Requested	Invisible	Fatal
Execution terminates	No	No ¹	No	Yes
Software is oblivious	No	No	Yes	Yes ²
Handled by environment	No	Yes	Yes	Yes

2.4. Stem content

The `:stem: latexmath` setting makes use of asciidoctor-mathematical for asciidoctor-pdf output.

Asciidoctor Mathematical is a Ruby gem that uses native extensions. It has a few system prerequisites which limit installation to Linux and macOS. Please refer to the [README in the RISC-V docs-templates repo](#) for information on the asciidoctor-mathematical install.

```
[stem]
++++
sqrt(4) = 2
++++
```

$$\sqrt{4} = 2$$

In some cases, you might want to make use of unicode characters. Keep in mind that asciidoctor-pdf currently only supports decimal character references. See [github.com/asciidoctor/asciidoctor-pdf/issues/486](#)

Hexadecimal unicode looks like it has problems in the pdf. This is gnarley.

Updates to asciidoctor-pdf: [github.com/asciidoctor/asciidoctor-pdf](#)

Chapter 3. Tables, symbols, and graphics

AsciiDoc makes standard tables easy and also supports the creation of complex tables.



Never use automated wrapping for table titles, figure captions, and example captions. AsciiDoctor reads a hard return as an indicator to start a new "Normal" paragraph.

3.1. More on examples

AsciiDoc tables can also be created directly from CSV data. Just set the format block attribute to CSV and insert CSV data inside the block delimiters directly:

```
[%header,format=csv]
|===
Artist,Track,Genre
Baauer,Harlem Shake,Hip Hop
The Lumineers,Ho Hey,Folk Rock
|===
```

The above renders as follows:

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop
The Lumineers	Ho Hey	Folk Rock

There are numerous formatting options available. While some of the property settings are cryptic, they can be quite useful. There are numerous examples available at asciidoc.org/newtables.html. Here one example of what can be done with spans alignment in tables from that page:

```
[cols="e,m,^,>s",width="25%"]
|=====
|1 >s|2 |3 |4
^|5 2.2+^.^|6 .3+<.>m|7
^|8
|9 2+>|10
|=====
```

Which renders as follows:

1	2	3	4
5	6		
8			
9		10	7

Following is code for a numbered encoding table with link target.



Annotations have been added to the code to illustrate their use.

```
[[proposed-16bit-encodings-1] ①
.proposed 16-bit encodings-1 ②
[width="100%",options=header]
|===
|15 |14 |13 |12 |11 |10 |9 |8 |7 |6 |5 |4 |3 |2 |1 |0 |instruction
3+|100|1|0|0|0 2+|field|0 |0 2+|00 | field 2+|00|mnemonic1
3+|100|1|0|0 3+|field|bit|1 3+|field 2+|00|mnemonic2
3+|110|1|0|0 3+|field|1 |0 3+|field 2+|00|mnemonic3
17+|This row spans the whole table
3+|100|1|1|1 8+| field 2+| 00 | mnemonic4
|===
```

- 1. Link target.
- 2. Numbered table title.

Table 2. proposed 16-bit encodings-1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	instr ucti on
100			1	0	0	0	field		0	0	00		field	00		mne mon ic1
100			1	0	0	field			bit	1	field			00		mne mon ic2
110			1	0	0	field			1	0	field			00		mne mon ic3
This row spans the whole table																
100			1	1	1	field								00		mne mon ic4

3.2. Unicode symbols

For pdf, five-digit unidole symbols generally don't work and some other unicode symbols are buggy. This happens because the Ruby asciidcotor-pdf toolchain makes ue of Prawn to build pdfs and it's Prawn that has the problems.

Here are a few unicode examples from en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references that might be useful:

As an example, \diamond is encoded as follows:

```
&#9830;
```

Table 3. Useful unicode for specifications

sym	num	name
\wedge	94	caret
\square	136	
\vdots	8942	vdots
\diamond	9830	name
"	0034	name
w	0077	w
\therefore	8756	therefore
#	9839	sharp
\mathbb{W}	1096	shcy
ϖ	982	piv varpi
ω	969	omega
\wp	8472	weierp wp
Σ	8721	sum
∞	8734	infin
\int	8747	integral
\neq	8800	not equal to
\leq	8804	le
\geq	8805	ge
\approx	8776	numerical approximation
D	68	mathematical D?
\Rightarrow	8658	rightwards double arrow
X	88	Latin Capital x
χ	967	Greek x
\times	215	times
$\boxed{\checkmark}$	9745	boxed checkmark
r	114	latin small letter r

For many symbols, then, we must turn to asciidoctor-mathematical. See [\[Superscripts and other mathematical notations\]](#).

Table 4. Unicode identified as not working

sym	num	name
□	9084	angzarr not working
□	8921	ggg not working
□	8617	hookleftarrow not working
□	9083	not checkmark not working

3.3. Graphics

While asciidoc can render graphics in all popular formats, by far the highest quality graphics rendering is from .svg format.

[WaveDrom sequence diagrams](#) are essential to the RISC-V specifications. We are in the process of phasing in an automated process for incorporating WaveDrom diagrams into the professional quality pdf output so please stay tuned.

[Asciidoctor-pdf](#) enables automation of diagrams from scripts, including WaveDrom.

Even as we are using WaveDrom to simplify the creation of accurate svgs for register diagrams, the graphical elements—those for the various diagrams—add complexity to the build.

3.3.1. Automated diagramming

Automated diagramming isn't easy. However, it can be important because some specifications require numerous, very accurate diagrams. Please feel free to ask questions about some of the details in this section.

The [asciidoctor-diagram extension](#) supports numerous diagram types, including [WaveDrom](#) diagramming for sequence and waveform diagrams, and [Graphviz digrams](#), which can be useful for illustrating software, networking, and security concepts.

The requirements for building WaveDrom diagrams are specified in the [docs templates README](#). Graphviz does not require anything in addition to asciidoctor-diagram.



To ensure findability of graphics source files, store the files in filetype subdirectories within the `images` directory.

ALERT: The build process for automated diagramming places an image file for each diagram into the `images` folder. Please do **not** check any generated image files into the git repository.

3.3.2. Wavedrom diagrams in specifications

The following json-formatted script, when added within an asciidoc block with the macro indicators `[wavedrom, ,]`, will embed the diagram output into the pdf:


```
{reg: [
  { bits: 7, name: 0x3b, attr: ['OP-32'] },
  { bits: 5, name: 'rd' },
  { bits: 3, name: 0x0, attr: ['ADD.UW'] },
  { bits: 5, name: 'rs1' },
  { bits: 5, name: 'rs2' },
  { bits: 7, name: 0x04, attr: ['ADD.UW'] },
]}
```



The macro `[wavedrom, ,]` has two commas and leaves a blank space as an implicit indicator to the build processor to auto-generate identifiers for the images after they are created. After the first comma you can insert a name as an identifier of the file that is created in the `/images` directory for embedding in the pdf.

DO NOT make the mistake of simply using `[wavedrom,svg]`. Without the second comma, the build interprets 'svg' as a target name because it is the second value within the macro.

In the specifications, there are numerous instances in which several Wavedrom diagrams are grouped and presented as a single figure. To handle these while preserving consistency in how the build renders figure titles, we are making use of a minimalistic, white graphic with the filename `image_placeholder.png` that blends in with the page background. Following is the pattern of its use (minus the macro indicator `[]` in the first line):

```
include::images/wavedrom/instruction_formats.adoc
[[instruction_formats]]
.Test for wavedrom
image::image_placeholder.png[]
```

With the following result:

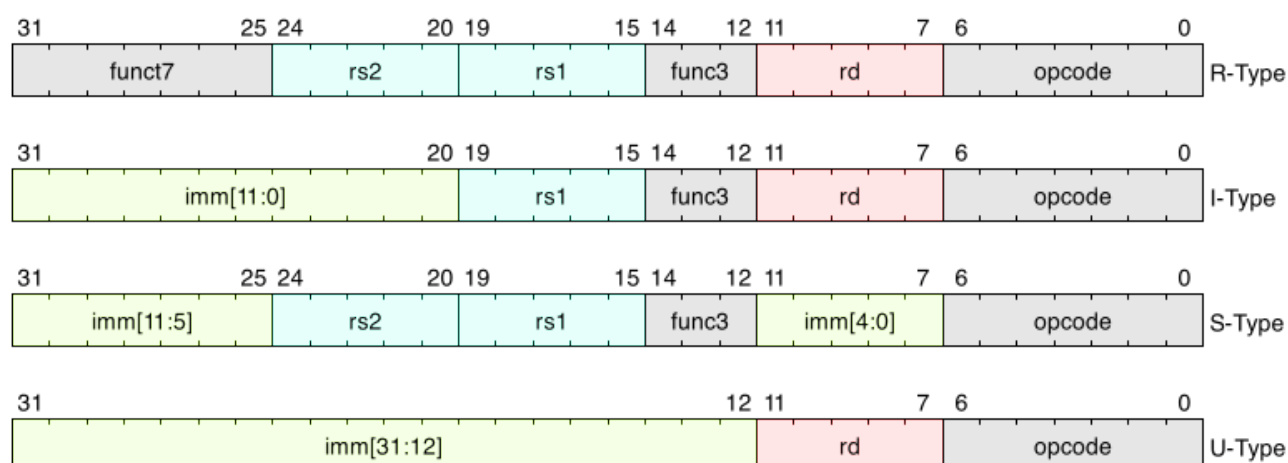


Figure 1. Test for wavedrom

1. Wavedrom code for all diagrams in this illustration stored in `wavedrom` subdirectory within the `images` directory.
2. Link target that, along with settings in the `book_header` file, automates the inclusion, in the text,



The following string is lacking macro brackets (`\[]`) that should appear after `filename.adoc` because adding the brackets will cause the include to activate even though it's within a code block. Best practice for automated diagramming is to save AsciiDoc files containing properly formatted AsciiDoc blocks, each block containing the code or script for either a single diagram or a group of diagrams that are presented together as a single figure.

```
include::images/wavedrom/filename.adoc
```

3.3.3. Graphviz

The Unpriv appendices contain Graphviz diagrams with associated keys that are arranged in tables. While in the LaTeX version, the diagrams and tables are arranged side-by-side, for the AsciiDoc version;

- each Graphviz diagram should be directly above the key table.
- store scripts for Graphviz diagrams in the images/graphviz directory, as <filename>.txt
- import the Graphviz by reference using the pattern in the following example.

```
.Sample litmus test
graphviz::images/graphviz/litmus_sample.txt[align="center"]

[cols="2,1"]
_Key for sample litmus test_
[width="60%",cols="^,<,<,<",options="header",align="center"]
|===
|Hart 0 | |Hart 1 |
| | $\vdots$  | | $\vdots$ 
| |li t1,1 | |li t4,4
|(a) |sw t1,0(s0) |(e) |sw t4,0(s0)
| | $\vdots$  | | $\vdots$ 
| |li t2,2 | |
|(b) |sw t2,0(s0) | |
| | $\vdots$  | | $\vdots$ 
|(c) |lw a0,0(s0) | |
| | $\vdots$  | | $\vdots$ 
| |li t3,3 | |li t5,5
|(d) |sw t3,0(s0) |(f) |sw t5,0(s0)
| | $\vdots$  | | $\vdots$ 
|===
```

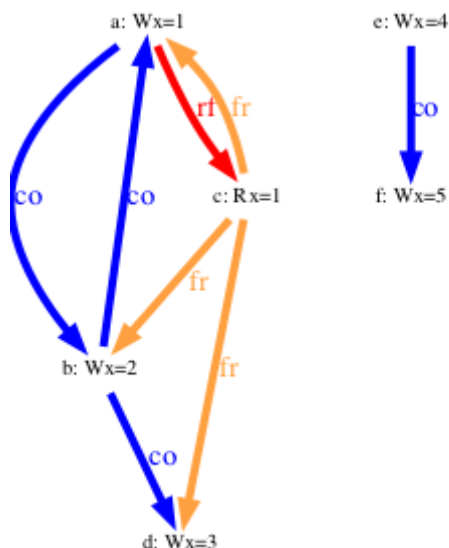


Figure 5. Sample litmus test

Key for sample litmus test

Hart 0		Hart 1	
	:		:
	li t1,1		li t4,4
(a)	sw t1,0(s0)	(e)	sw t4,0(s0)
	:		:
	li t2,2		
(b)	sw t2,0(s0)		
	:		:
(c)	lw a0,0(s0)		
	:		:
	li t3,3		li t5,5
(d)	sw t3,0(s0)	(f)	sw t5,0(s0)
	:		:



The procedures for Graphviz diagrams are similar but not identical to procedures for [Section 3.3.2, “Wavedrom diagrams in specifications”](#).

Following is an example of Graphviz diagram source:

```
.Graphviz s
[graphviz, target="ethane",svg]
....
graph ethane {
    C_0 -- H_0 [type=s];
    C_0 -- H_1 [type=s];
    C_0 -- H_2 [type=s];
    C_0 -- C_1 [type=s];
    C_1 -- H_3 [type=s];
    C_1 -- H_4 [type=s];
    C_1 -- H_5 [type=s];
}
....
```

This renders as:

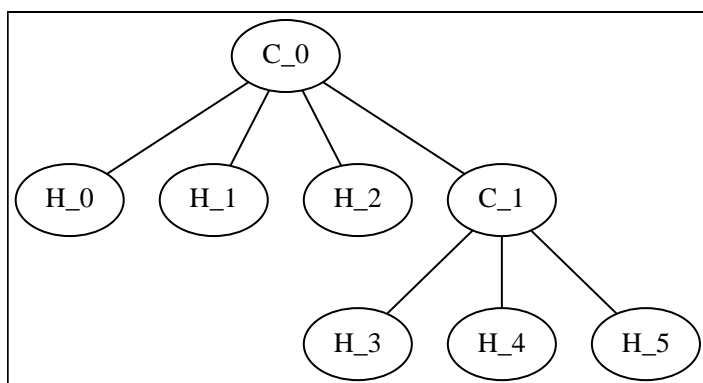


Figure 6. Graphviz s

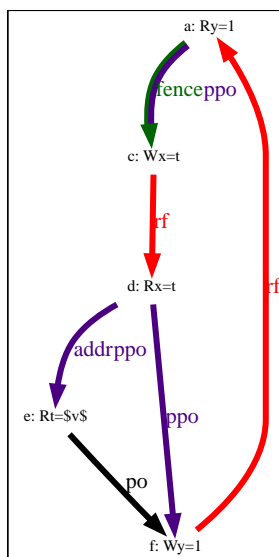


Figure 7. An example graphviz diagram from a specification

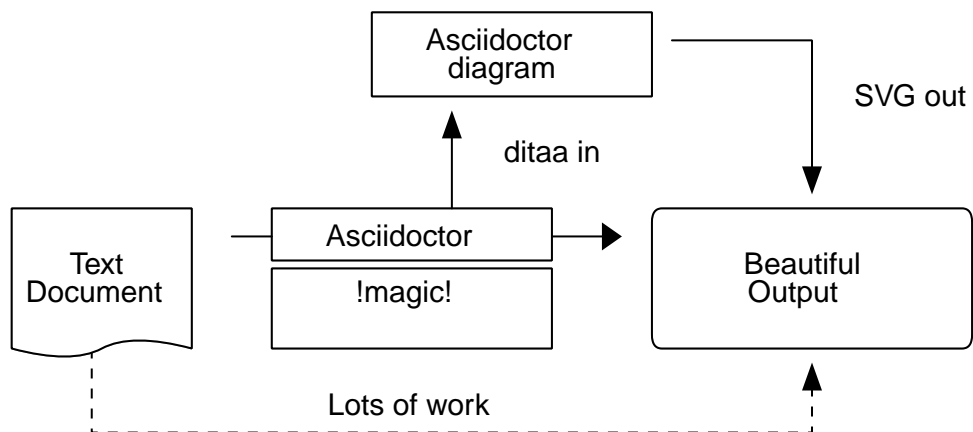
3.3.4. Additional diagram type examples

Following is source for simple ditaa diagram:

```
[ditaa,target="image-example",svg]
....

      +-----+
      | AsciiDoctor |-----+
      |  diagram   |         |
      +-----+         | SVG out
      ^                 |
      | ditaa in        |
      |                 |
      |                 v
+-----+ +-----+ /-----\
|      | --+ AsciiDoctor +--> |      | | |
| Text | +-----+         | Beautiful |
|Document| | !magic! |         | Output  |
| {d}| |         |         |
+---+---+ +-----+ \-----/
:
|           Lots of work           |
+-----+
....
```

Which renders to:

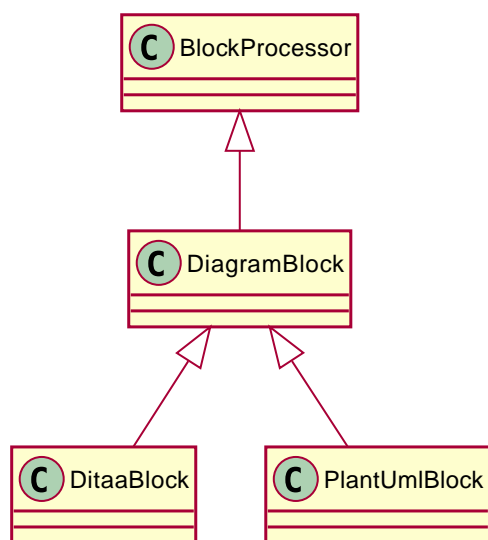


Following is source for a simple plantuml diagram:

```
[plantuml, diagram-classes, svg]
....
class BlockProcessor
class DiagramBlock
class Ditaablock
class PlantUmlBlock

BlockProcessor <|-- DiagramBlock
DiagramBlock <|-- Ditaablock
DiagramBlock <|-- PlantUmlBlock
....
```

Which renders to:



AsciiDoctor supports additional diagram types. For information on additional diagram types, see the [AsciiDoctor-diagram documentation](#).

A mention of Bytefeild

Currently the idea of making use of Bytefeild as an additional diagram type is being explored.

3.4. Mathematical notations



AsciiDoctor-mathematical has some limitations. For inline expressions, the graphical representations appear small and they are centered vertically. In some cases where there is a single-character AsciiDoctor-mathematical expression, it unintentionally looks like a superscript. For this reason, always use viable alternatives like italics or unicode (see [Section 3.2, “Unicode symbols”](#)).

3.4.1. Superscripts and subscripts

To indicate a superscript, enclose the string for the superscript in carets as in the following example:

2^8

Which renders as:

 2^8

You can indicate text in a superscript as well:

 1234^{NOTE}

Which renders as:

 1234^{NOTE}

For subscripts, use tildes:

 $C_2 H_6$

With the following result:

 $C_2 H_6$

An example:

```
"Well the H~2~O formula written on their whiteboard could be part
of a shopping list, but I don't think the local bodega sells
E=mc^2^," Lazarus replied.
```

Renders as:

"Well the H₂O formula written on their whiteboard could be part of a shopping list, but I don't think the local bodega sells E=mc²," Lazarus replied.

3.4.2. Latexmath

You can make use of LaTeX notation as in the following:

$$C = \alpha + \beta Y^{\gamma} + \epsilon$$

Which renders as:

$$C = \alpha + \beta Y^{\gamma} + \epsilon$$



Latexmath rendering has some limitations with respect to sizing and placement inline. This happens because of how the images for the mathematical symbols are rendered within the build process. For this reason, please avoid using single character latexmath expresions inline and prefentially make use of unicode or superscripts and subscripts when possible.

Chapter 4. Blocks, notes and markers

RISC-V specifications are notable for their extended commentaries that explain the thinking behind various important aspects of the technologies.

In most cases, contributors should make use of admonition blocks for their commentaries.

4.1. Blocks

Asciidoctor allows for many types of blocks, as documented at docs.asciidoctor.org/asciidoc/latest/blocks/.

4.1.1. Sidebars

Sidebars provide for a form of commentary.

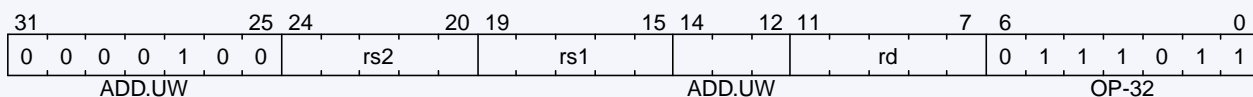
```
****
This is content in a sidebar block.

image:example-3.svg[]

This is more content in the sidebar block.
****
```

This renders as follows:

This is content in a sidebar block.



This is more content in the sidebar block.

You can add a title, along with any kind of content. Best practice for many of the "commentaries" in the LaTeX source that elucidate the decisionmaking process is to convert to this format with the **TIP** icon that illustrates a conversation or discussion, as in the following example:

.Optional Title

Sidebars are used to visually separate auxiliary bits of content that supplement the main text.

TIP: They can contain any type of content, including admonitions like this, and code examples like the following.

.Source code block within a sidebar

[source,js]

/---- (1)

```
const { expect, expectCalledWith, heredoc } = require('../test/test-utils')
```

/---- (2)

1 and 2. Escapes are necessary to preserve this as an AsciiDoc code example.

Once the escapes are removed, the above renders with both the admonition and code blocks within the sidebar:

Optional Title

Sidebars are used to visually separate auxiliary bits of content that supplement the main text.



They can contain any type of content, including admonitions like this, and code examples like the following.

Listing 1. Source code block in a sidebar

```
const { expect, expectCalledWith, heredoc } = require('../test/test-utils')
```

4.1.2. Admonition blocks

Five kinds of standard admonition blocks are available in asciidoc and these can be mapped to either default or custom icons.

```
[NOTE]
```

```
====
```

```
This is an example of an admonition block.
```

```
Unlike an admonition paragraph, it may contain any AsciiDoc content.
```

```
The style can be any one of the admonition labels:
```

```
* NOTE
```

```
* TIP
```

```
* WARNING
```

```
* CAUTION
```

```
* IMPORTANT
```

```
====
```

This renders as:



This is an example of an admonition block.

Unlike an admonition paragraph, it may contain any AsciiDoc content. The style can be any one of the admonition labels:

- *NOTE*
- *TIP*
- *WARNING*
- *CAUTION*
- *IMPORTANT*

For a single paragraph admonition, simply use a double colon:

```
NOTE: Note content.
```

which renders as:



Note content.

Alternate octicons:

- alert-24
- comment-discussion-24
- flame-24
- info-24
- pencil-24
- question-24
- sheild-24

- squirrel-24
- zap-24

Another example of an admonition block:

```
[IMPORTANT]
.Feeding the Werewolves
====
While werewolves are hardy community members, keep in mind the following dietary
concerns:

. They are allergic to cinnamon.
. More than two glasses of orange juice in 24 hours makes them howl in harmony
with alarms and sirens.
. Celery makes them sad.
=====
```

Rendered:



Feeding the Werewolves

While werewolves are hardy community members, keep in mind the following dietary concerns:

- 1. They are allergic to cinnamon.*
- 2. More than two glasses of orange juice in 24 hours makes them howl in harmony with alarms and sirens.*
- 3. Celery makes them sad.*

github.com/asciidocctor/asciidocctor-pdf/blob/master/docs/theming-guide.adoc#key-prefix-admonition-icon

The default admonition icons don't look right for RISC-V specification, and alternate icons and colors have been set in `risc-v_spec-pdf.yml`. and will be considered.

Current icons, edited to tone down color:



note



tip



warning



caution



important

Table 5. Customized colors for icons

Icon	default	customized
NOTE	19407c	6489b3
TIP	111111	5g27ag
WARNING	bf6900	9c4d4b
CAUTION	bf3400	c99a2c
IMPORTANT	bf0000	b58f5b

4.1.3. Code blocks

AsciiDoc enables code blocks that support syntax highlighting.

For example, preceding a block with a macro `[source, json]` enables `json` syntax highlighting:

```
{
  "weather": {
    "city":      "Zurich",
    "temperature": 25,
  }
}
```

While syntax highlighters for machine code that integrate with the Asciidoctor Ruby toolchain do leave something to be desired, the Rouge highlighter enables line numbers within the code examples.

We are numbering examples as in the following:

```
.A spinlock with fences
[source%linenums, asm]
....
        sd          x1, (a1)      # Arbitrary unrelated store
        ld          x2, (a2)      # Arbitrary unrelated load
        li          t0, 1         # Initialize swap value.
again:
        amoswap.w    t0, t0, (a0)  # Attempt to acquire lock.
        fence        r, rw        # Enforce "acquire" memory ordering
        bnez         t0, again     # Retry if held.
        # ...
        # Critical section.
        # ...
        fence        rw, w        # Enforce "release" memory ordering
        amoswap.w    x0, x0, (a0)  # Release lock by storing 0.
        sd          x3, (a3)      # Arbitrary unrelated store
        ld          x4, (a4)      # Arbitrary unrelated load
....
```

With the following result:

Listing 2. A spinlock with fences

```

sd      x1, (a1)    # Arbitrary unrelated store
ld      x2, (a2)    # Arbitrary unrelated load
li      t0, 1       # Initialize swap value.
again:
  amoswap.w t0, t0, (a0) # Attempt to acquire lock.
  fence    r, rw       # Enforce "acquire" memory ordering
  bnez     t0, again   # Retry if held.
  # ...
  # Critical section.
  # ...
  fence    rw, w       # Enforce "release" memory ordering
  amoswap.w x0, x0, (a0) # Release lock by storing 0.
  sd      x3, (a3)    # Arbitrary unrelated store
  ld      x4, (a4)    # Arbitrary unrelated load

```

4.2. Change bars

Change indicators within text files are exceedingly useful and also can be equally complex to implement. Please consider the fact that much of the software programming for Git revolves around handling various kinds of change indicators.

In exploring possible implementation of change bars for RISC-V, we have looked for a solution that is as simple as possible while maximizing value with respect to the time invested in implementing, maintaining, and using the tools and procedures.

The suggested solution makes use of:

- an AsciiDoc `role`.
- modification of two files in the Ruby gem with code snippets (see procedure in the README for github.com/riscv/docs-templates).
- Git features.
- a few procedures associated, specifically, with Git updates.

4.2.1. Indicate changes

With apologies for requiring a manual step at this time, indicators for the changed lines must be inserted:

```

[.Changed]#SELECT clause#

Text without the change bar

[.Changed]#Text with the change bar#

```

SELECT clause

Text without the change bar

Text with the change bar

For change bars associated with headings, place the change indicator after the heading indicator and before the text, like the following:

```
== [.Changed]#SELECT clause#
```

4.2.2. Check for changed lines before a `git commit`

You can double check for all changed lines just before doing a commit, using this pattern:

```
git blame <file> | grep -n '^0\{8\}' | cut -f1 -d:
```

This lists the line numbers of changes within the specified file like the following:

```
5
38
109
237
```

4.3. Footnotes

Asciidoc has a limitation in that footnotes appear at the end of each chapter. Asciidoctor does not support footnotes appearing at the bottom of each page.

You can add footnotes to your presentation using the footnote macro. If you plan to reference a footnote more than once, use the footnote macro with a target that you identify in the brackets.

```
Initiate the hail-and-rainbow protocol at one of three levels:

- doublefootnote:[The double hail-and-rainbow level makes my toes tingle.]
- tertiary
- apocalyptic

A bold statement!footnote:disclaimer[Opinions are my own.]

Another outrageous statement.footnote:disclaimer[]
```

Renders as:

The hail-and-rainbow protocol can be initiated at three levels:

- double ^[1]
- tertiary

- apocalyptic

A bold statement! ^[2]

Another outrageous statement.^[2]

4.4. Index markers

There are two types of index terms in AsciiDoc:

A flow index term. appears in the flow of text (a visible term) and in the index. This type of index term can only be used to define a primary entry:

```
indexterm2:[<primary>] or ((<primary>))
```

A concealed index term. a group of index terms that appear only in the index. This type of index term can be used to define a primary entry as well as optional secondary and tertiary entries:

```
indexterm:[<primary>, <secondary>, <tertiary>]
```

--or--

```
(((<primary>, <secondary>, <tertiary>)))
```

```
The Lady of the Lake, her arm clad in the purest shimmering samite,
held aloft Excalibur from the bosom of the water,
signifying by divine providence that I, ((Arthur)), ❶
was to carry Excalibur (((Sword, Broadsword, Excalibur))). ❷
That is why I am your king. Shut up! Will you shut up?!
Burn her anyway! I'm not a witch.
Look, my liege! We found them.
```

```
indexterm2:[Lancelot] was one of the Knights of the Round Table. ❸
indexterm:[knight, Knight of the Round Table, Lancelot] ❹
```

- ❶ The double parenthesis form adds a primary index term and includes the term in the generated output.
- ❷ The triple parenthesis form allows for an optional second and third index term and does not include the terms in the generated output (a concealed index term).
- ❸ The inline macro `indexterm2\[primary]` is equivalent to the double parenthesis form.
- ❹ The inline macro `indexterm\[primary, secondary, tertiary]` is equivalent to the triple parenthesis form.

If you're defining a concealed index term (the `indexterm` macro), and one of the terms contains a comma, you must surround that segment in double quotes so the comma is treated as content. For

example:

```
I, King Arthur.
indexterm:[knight, "Arthur, King"]
```

I, King Arthur.

--or--

```
I, King Arthur.
(((knight, "Arthur, King")))
```

I, King Arthur.

4.5. Bibliography and references

There are two ways of handling bibliographies:

- making manual entries to which you can create links from the text in the body of your document.
- using automated features provided by `asciidoctor-bibtex`[]

You can add bibliographic entries to the last appendix that you use in a book document.

4.5.1. Automated bibliography procedures with `asciidoctor-bibtex`

`Asciidoctor-bibtex` enables options that allow for establishing a single source of bibliographic entries that we can use for RISC-V specifications. As an added benefit we can make use of existing `bibtex` files.

For `asciidoctor-bibtex` to work, please install the Ruby gems as documented in the docs-templates README file.



This has now been tested and is the preferred procedure for adding a bibliography.

The doc header file in the docs-templates repo now contains the following attributes for the purpose of implementing a bibliography using `asciidoctor-bibtex`:

```
:bibtex-file: resources/references.bib
:bibtex-order: alphabetical
:bibtex-style: apa
```

The repo also contains the most recent version of the `riscv-spec.bib` file for `asciidoctor-bibtex` to use while building the bibliography.

When you run `asciidoctor-bibtex` as part of the build, it searches for the `bibtex` file first in the folder and subfolders of the document header, and then in `~/Documents`.

Within your text, add author-year references using the pattern:

```
cite:[riscvtr(12)]
```

with the result, ([Waterman et al., 2011, p. 12](#))

Add age numbers (locators) using the pattern:

```
cite:[Kim-micro2005(45)]
```

with the result: ([Kim et al., 2005, p. 45](#))

Add pretext using the pattern:

```
cite:See[Kim-micro2005(45)]
```

with the result: (See [Kim et al., 2005, p. 45](#))

It's possible to include other files, which are also processed.



To to prevent problems with other appendices, leave keep the index as the second-to-last appendix and the bibliography as the last appendix in you list of included chapter sections within the book-header file.

Citations must be contained within a single line.

The bibliography section of the book must be set up as follows, to receive the entries during the build:

```
== Bibliography

bibliography:: []
```



*When using the automated option, do not manually add entries to the **bibliography.adoc** file.*

Following are example json-formatted bibliographic entries:

```

@book{Lane12a,
  author = {P. Lane},
  title = {Book title},
  publisher = {Publisher},
  year = {2000}
}

@book{Lane12b,
  author = {K. Mane and D. Smith},
  title = {Book title},
  publisher = {Publisher},
  year = {2000}
}

@article{Anderson04,
  author = {J. R. Anderson and D. Bothell and M. D. Byrne and S. Douglass and C.
Lebiere and Y. L. Qin},
  title = {An integrated theory of the mind},
  journal = {Psychological Review},
  volume = {111},
  number = {4},
  pages = {1036--1060},
  year = {2004}
}

```

4.5.2. Manual bibliography procedures

While the automated procedure and use of the RISC-V bibtex file is preferred, it is also possible to manually create and reference a bibliography.

Text with markup that will generate links:

```

_The Pragmatic Programmer_ <<pp>> should be required reading for all developers.
To learn all about design patterns, refer to the book by the "`Gang of Four`"
<<gof>>.

```

Links from within text to bibliographic entries:

```

[[bibliography]]
== References

* [[[pp]]] Andy Hunt & Dave Thomas. The Pragmatic Programmer:
From Journeyman to Master. Addison-Wesley. 1999.
* [[[gof,gang]]] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides.
Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
1994.

```

Text that links to bibliography:

```
_The Pragmatic Programmer_ <<pp>> should be required reading for all developers.  
To learn all about design patterns, refer to the book by the "`Gang of Four`"  
<<gof>>.
```

[1] The double hail-and-rainbow level makes my toes tingle.

[2] Opinions are my own.

Chapter 5. Editing Wavedrom diagrams for Unpriv



This is a work in progress. I am at this point not versed in content issues with respect to the diagrams and in addition I am discovering the editing process through trial and error. There are properties for which I have not yet tested procedures. For that reason I welcome helpful suggestions and critiques. Please email help@riscv.org.

5.1. Relevant contextual information

Wavedrom is a utility that is available at wavedrom.com/.

The Wavedrom diagrams used in the Unpriv specification include opcodes and related information. The diagrams are vital to understanding the specification.

For some RISC-V specifications, Wavedrom diagrams have been placed within topics that are seen as [information objects](#) that have identifiable patterns but have yet to be fully and formally defined.

The Wavedrom diagrams for the Unpriv specification are contained in the src/images/wavedrom folder in the [convert2adoc](#) branch.

Wavedrom diagrams are coded using a javascript-like code structure.

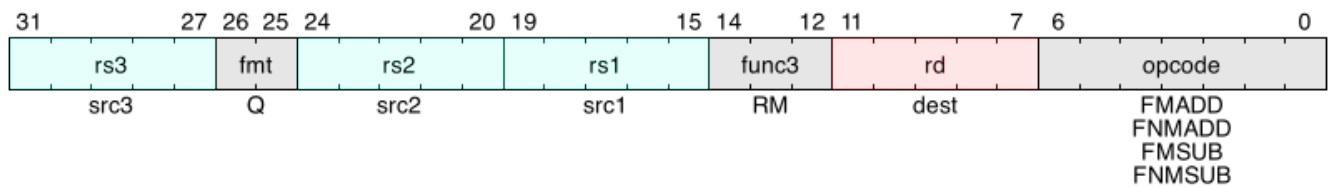
The working assumption is that we need to start with a 1:1 converted version, with the exception that we are adding missing figure titles, table titles, and example titles for consistency in the specification.

5.2. Example Wavedrom code, before and after

Following is an example Wavedrom file that is typical of one the needs just a few edits, minus the `[]` brackets that indicate a macro (because using the macro even within a code block activates a process in the AsciiDoctor build):

```
{reg: [
  {bits: 7, name: 'opcode', attr: ['FMADD', 'FNMADD', 'FMSUB', 'FNMSUB'],
type: 8},
  {bits: 5, name: 'rd', attr: 'dest', type: 2},
  {bits: 3, name: 'func3', attr: 'RM', type: 8},
  {bits: 5, name: 'rs1', attr: 'src1', type: 4},
  {bits: 5, name: 'rs2', attr: 'src2', type: 4},
  {bits: 2, name: 'fmt', attr: 'Q', type: 8},
  {bits: 5, name: 'rs3', attr: 'src3', type: 4},
]}
```

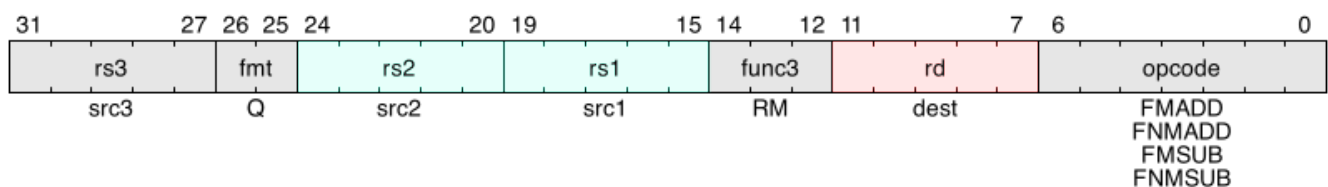
This renders as follows:



Listing 3. For convenience, it makes sense to line up the `type` attribute so that it remains easy to see:

```
{reg: [
  {bits: 7, name: 'opcode', type: 8, attr: ['FMADD', 'FNMADD', 'FMSUB',
'FNMSUB'], },
  {bits: 5, name: 'rd', type: 2, attr: 'dest', },
  {bits: 3, name: 'func3', type: 8, attr: 'RM', },
  {bits: 5, name: 'rs1', type: 4, attr: 'src1', },
  {bits: 5, name: 'rs2', type: 4, attr: 'src2', },
  {bits: 2, name: 'fmt', type: 8, attr: 'Q', },
  {bits: 5, name: 'rs3', type: 8, attr: 'src3', },
]}
```

The output remains the same:



2. For each line that contain a single value for the `attr` attribute:

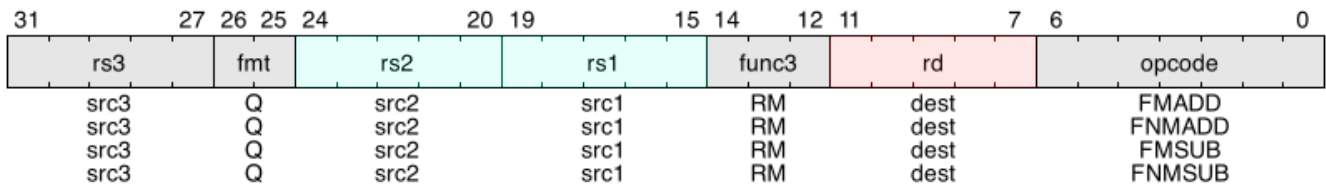
- add `[]` to contain additional values—and--
- follow the convention for commas to contain and separate additional values until all the lines contain the same number of values for `attr` that are in the 'opcodes' row:

```
{reg: [
  {bits: 7, name: 'opcode', type: 8, attr: ['FMADD', 'FNMADD', 'FMSUB',
'FNMSUB'], },
  {bits: 5, name: 'rd', type: 2, attr: ['dest','dest','dest','dest',], },
  {bits: 3, name: 'func3', type: 8, attr: ['RM','RM','RM','RM',], },
  {bits: 5, name: 'rs1', type: 4, attr: ['src1','src1','src1','src1',], },
  {bits: 5, name: 'rs2', type: 4, attr: ['src2','src2', 'src2', 'src2', ],
},
  {bits: 2, name: 'fmt', type: 8, attr: ['Q','Q','Q','Q',], },
  {bits: 5, name: 'rs3', type: 8, attr: ['src3','src3','src3','src3',], },
]}
```



Wavedrom makes use of straight single quotes like `' '` rather than diagonal single quotes like `` ``.

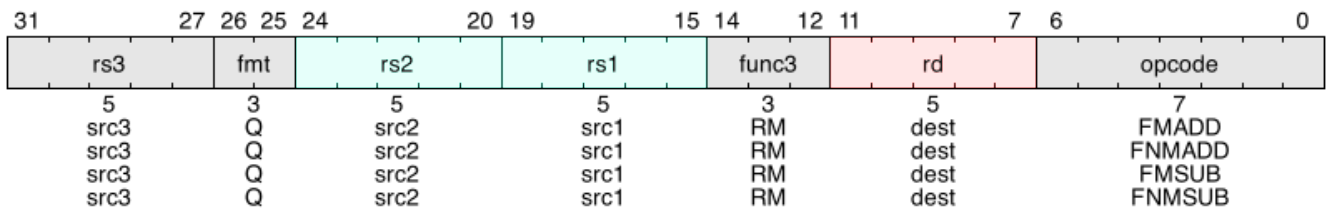
Here's the result:



3. Check the LaTeX version of the diagram to check for the numerical values that are needed within the missing row:

```
{reg: [
  {bits: 7, name: 'opcode', type: 8, attr: ['8', 'FMADD', 'FNMADD', 'FMSUB',
'FNMSUB'], },
  {bits: 5, name: 'rd', type: 2, attr: ['6', 'dest','dest','dest','dest'],},
},
  {bits: 3, name: 'func3', type: 8, attr: ['4', 'RM','RM','RM','RM'], },
  {bits: 5, name: 'rs1', type: 4, attr: ['6', 'src1','src1','src1','src1'],},
},
  {bits: 5, name: 'rs2', type: 4, attr: ['6', 'src2','src2', 'src2', 'src2',
], },
  {bits: 2, name: 'fmt', type: 8, attr: ['4', 'Q','Q','Q','Q'], },
  {bits: 5, name: 'rs3', type: 8, attr: ['6', 'src3','src3','src3','src3'],},
},
]}
```

Now the diagram should contain all of the content that exists within the LaTeX version:



4. If you or a member of your team can build locally, please ensure that someone checks that the diagrams build without errors.
5. Generate a PR to the convert2adoc branch and indicate whether you or a member of the team has tested your changes in a local build.
6. As always, thanks for your participation in the success of RISC-V.

5.3. Caveats for editing wavedrom diagrams

At the time of this writing, we have noticed the following unexpected results during diagram builds using the asciidoctor-pdf toolchain, as follows:

- Some, but not all, unicode that works in AsciiDoc (see [Table 3, “Useful unicode for specifications”](#)) actually breaks the Wavedrom diagram build, and other unicode does not break the Wavedrom diagram build but still doesn’t render properly.
- Latexmath appears to not work at all in Wavedrom diagrams.
- After struggling to understand why various options that we explored for an acceptable ≠ in

Wavedrom diagrams and discovering the above rather confusing results, we decided to use != as a workaround. With the fact that both Ascoddoctor and Wavedrom are evolving, and also the fact that bytearray is being considered as an alternative diagrams rendering solution, it seems possible that this workaround will be temporary.

Chapter 6. Write simply

All professional technical writers eventually learn to keep their writing as simple as possible.

6.1. Writing is rewriting

No matter how accomplished the writer, a first draft is a first draft.

- In high tech industries, writers often allow publishing of a first draft in order to provide users with the information that they need.
- In the entertainment industry, even a script for which a company paid millions might undergo more than one "page one rewrite" before production.

The above seems counter-intuitive.

Here are some writing guidelines:

- Organize information according to a pattern that supports one of both of the following:
 - Phases of development from idea through development, testing, and deployment.
 - Concepts, supporting a learning process, from simple and easy to complex and difficult.
 - Prioritize documenting information that all users require.
 - If possible, illustrate decisions that users must make with examples that show potential outcomes.
- As much as possible, maintain parallelism with respect to how you handle information.

6.2. Develop style guidelines only as needed

While we develop style guidelines, this chapter serves for collecting style guidelines specific to RISC-V. At this point (December 2021) there is almost no content.

Please feel free to suggest ideas and content for the RISC-V style guidelines, and please don't think that you need to add polished content.

Small organizations like RISC-V usually identify existing style guides and then document the "deltas," including cases where it makes sense to deviate from what's in the existing style guides and also what needs to be added.

I have a list of existing guides that we can use as a starting point. These are likely the most pertinent existing style guidelines that were created for addressing the needs of highly technical audiences as opposed to consumer end user audiences:

- [Write the Docs style guide](#)
- [Google's developer docs style guide](#)
- [Stylepedia](#)
- [NetApp style guide](#)



One of the typical problems that technical writers that specialize in developer-facing documentation tend to find is the misapplication of maxims that are only appropriate for addressing usability issues that arise with consumer end-user audiences. What works for one audience does not always work for all audiences. For example, I have found that developers tend to prefer long topics (this has been very consistent), while consumer end users tend to prefer short topics. Also developers often prefer code examples to graphical representations, while consumer end users are quite consistent in wanting graphical illustrations.

6.3. Active voice

Technical writers are always encouraged to use active voice as much as possible (irony fully intended).

When writing for highly technical audiences, strictly adhering to the use of active voice can result in convoluted sentences. As a result, while attempting to use active voice can result in improved clarity, contributors can choose to make use of passive voice. That said, we still encourage RISC-V contributors make use of active voice as much as possible.

6.4. Identify information types

While other style issues are important, addressing the need for guidelines on the information type(s) associated with instructions appears to be something that could increase the quality and efficiency of contributions to RISC-V specifications.

While documentation for most technologies require special style considerations, RISC-V's focus on ISAs (Instruction Set Architectures) means that its documentation contains information types for which we will be setting standards. RISC-V contributors are in the process of identifying how information about mnemonics can be clearly typed, named, and stored in a database that can then be used as a single source of truth. This is analogous to how some API reference information is identified, typed, and stored in databases.

Handling reference information in this way has many benefits that are easy to recognize. From the point of view of working with the Asciidoctor toolchain, each individual topic of this type can:

- be stored in a database
- contain attributes as needed
- make use of AsciiDoc Roles

6.5. RISC-V-specific style guidelines

This section is here to contain results of style discussions that have emerged from direct email, pull requests, issues, and other venues.

6.5.1. Table captions: above or below the table?

RE: A request for table captions to be placed below tables.

Discussion: A simple Google search on the topic of table caption placement resulted in all but one

online discussion stating that table captions should appear above tables. The single result that did not make this statement mentioned that while normal placement is above the table, there are occasional instances where organizations decide to place table captions below.

Index

K

knight

Arthur, King, [31](#), [31](#)

Bibliography

Kim, H., Mutlu, O., Stark, J., & Patt, Y. N. (2005). Wish Branches: Combining Conditional Branching and Predication for Adaptive Predicated Execution. *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, 43–54.

Waterman, A., Lee, Y., Patterson, D. A., & Asanović, K. (2011). *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA* (UCB/EECS-2011-62; Issue UCB/EECS-2011-62). EECS Department, University of California, Berkeley.

The image features the RISC-V logo in white on a dark grey background. The logo consists of a stylized 'R' icon followed by the text 'RISC-V'. The background is a blue-tinted, high-tech illustration of a circuit board with various components and glowing lines.

RISC-V