

# Authoring and Editing RISC-V Specifications

Version v, 2024-09-10:

## Table of Contents

Preface .....	1
1. Contributing to the RISC-V documentation .....	2
1.1. How to join RISC-V documentation .....	2
2. AsciiDoc authoring for RISC-V contributors .....	3
2.1. RISC-V AsciiDoc authoring assets .....	3
2.2. Writing .....	3
2.2.1. Text editors with AsciiDoc support .....	3
3. AsciiDoc basics .....	5
3.1. Paragraphs .....	5
3.1.1. Basics of blocks and indents .....	5
3.2. Headers .....	6
3.3. Lists .....	6
3.3.1. Unordered list .....	7
3.3.2. Ordered list .....	7
3.3.3. Nested list .....	7
3.3.4. Add a title to a list .....	8
3.4. Hyperlinks and cross references .....	9
3.4.1. Hyperlinks .....	9
3.4.2. Cross references .....	9
3.5. Tables .....	9
3.5.1. General rules for tables .....	10
3.5.2. Simple table .....	10
3.5.3. Adding table headers .....	10
3.5.4. Table captions .....	11
3.5.5. AsciiDoc tables from CSV data .....	12
3.6. Unicode symbols .....	13
3.7. Mathematical notations .....	15
3.7.1. Superscripts and subscripts .....	15
3.7.2. Latexmath .....	16
3.7.3. Stem content .....	16
3.8. Admonition blocks .....	16
3.8.1. Single paragraph admonition .....	18
3.8.2. Admonition blocks .....	18
3.8.3. Admonition with a title .....	18
3.8.4. RISC-V admonition icon colors .....	19
3.9. Code blocks .....	20
3.10. Change bars .....	21
3.10.1. Indicate changes .....	21
3.10.2. Check for changed lines before a <b>git commit</b> .....	22
3.11. Footnotes .....	22
3.12. Sidebars .....	23

<b>4. Index and bibliography</b>	<b>25</b>
4.1. Index markers	25
4.2. Bibliography and references	26
4.2.1. Creating an automated bibliography with asciidoctor-bibtex	26
4.2.2. Manual bibliography procedures (deprecated)	28
<b>5. Graphics</b>	<b>30</b>
5.1. Graphics best practices	30
5.2. Wavedrom diagrams in specifications	30
5.3. Explanation	32
5.4. Graphviz	33
5.5. Dita diagrams	35
5.6. Bytefield diagrams	37
5.7. Editing Wavedrom diagrams for Unpriv	38
5.7.1. Relevant contextual information	38
5.7.2. Example Wavedrom code, before and after	38
5.7.3. Caveats for editing wavedrom diagrams	41
<b>6. RISC-V Style guidelines</b>	<b>42</b>
6.1. Basic formatting	42
6.2. CSR formatting	42
6.3. Table formatting	43
6.3.1. Column header formatting	43
6.3.2. Punctuation in tables	43
<b>7. Best practices</b>	<b>45</b>
7.1. Use present tense	45
7.2. Use active voice	45
7.3. Use simple and direct language	45
7.4. Address the reader as "you"	45
7.5. Avoid Latin phrases	45
7.6. Avoid jargon and idioms	46
7.7. Avoid statements about the future	46
7.8. Avoid statements that will soon be out of date	46
7.9. Avoid words that assume a specific level of understanding	46
7.10. Other style guidelines	46
<b>8. Word usage</b>	<b>48</b>
<b>9. Linting</b>	<b>52</b>
9.1. Tooling	52
<b>Index</b>	<b>53</b>
<b>Bibliography</b>	<b>54</b>

## 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 communication of the specs.

AsciiDoc is currently the most feature-rich of the popular lightweight markup languages. As an Open Source effort, it is gaining wider adoption and there is an [AsciiDoc working group](#) to standardize the AsciiDoc specification as well as other ways to get involved.

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 [github.com/riscv/docs-spec-template](https://github.com/riscv/docs-spec-template) repo allows you to jump into writing a spec.

If you'd like to discuss the documentation build toolchain, please join the RISC-V group [sig-documentation](#) devoted to supporting the RISC-V spec documentation. 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. Contributing to the RISC-V documentation

Welcome to the RISC-V documentation guidelines. This document is maintained by the [RISC-V Doc Sig](#).

This group gathers and works issues from the [Doc-sig GitHub repository](#).

RISC-V documentation contributors:

- Improve existing content
- Create new content
- Ensure content follows the style guidelines
- Translate the documentation (coming soon!)

### 1.1. How to join RISC-V documentation

- Create a Linux foundation ID with [LFX](#).
- Become a [member of the RISC-V International](#) organization.

## Chapter 2. AsciiDoc authoring for RISC-V contributors

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

Asciidoctor toolchains:

- [asciidoctor](#), which is well-established and written in ruby.
- [antora](#), which is newer and written in javascript.
- Python-based, which is legacy and in maintenance mode.

### 2.1. RISC-V AsciiDoc authoring assets

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

The docs-templates repo also contains assets, such as 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 required 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 HTML output by using the following command:

```
asciidoctor book_header.adoc
```

## 2.2. Writing

To begin writing in AsciiDoc, select a text editor. Here's [Section 2.2.1](#).

For quick reference, see [AsciiDoc Syntax Quick Reference](#). Most of the markup for the specifications is simple and should have a familiar feel to those who have used Git-flavored Markdown. For RISC-V specifications, it is the procedures for [Chapter 5](#) that add complexity.

The [AsciiDoc Writers Guide](#) contains details about AsciiDoc markup.

Here are a few additional, useful links:

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

### 2.2.1. Text editors with AsciiDoc support

There isn't a true WYSIWYG editor for AsciiDoc. However, there are live preview options that are listed in the [Asciidoctor documentation](#) that can help you see what the output looks like. You can also use your favorite text editor, or perhaps switch to one that has good AsciiDoc linting.

The following list contains links to resources for text editors/IDEs support of AsciiDoc:

- Information on helpful [AsciiDoc tools that integrate with several popular IDEs](#)
- [vscode](#)
  - [asciidoctor-vscode extension](#)
- [vim](#) (supports asciidoc natively)
- [emacs](#)
  - [emacs Wiki AsciiDoc page](#)
    - [bbatsov/adoc-mode](#)
- [AsciidocFX](#)
- [IntelliJ IDEs](#) (NOTE: IntelliJ IDEs are not free/OSS)
  - [jetbrain asciidoc plugin](#)
- [Sublime Text](#) (NOTE: Sublime is not free)
  - [asciidoc package for Sublime Text2](#)

## Chapter 3. 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/Asciidoctor [writers' guide](#).
- AsciiDoc [quick reference](#).
- Asciidoctor [user manual](#).

In addition, you can ask questions in the [Asciidoctor discussion list](#).

As is true of any complex process, AsciiDoc/Asciidoctor has some quirks. Please be certain to use the templates, as the templates provide you with files that 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 requires the use of Ruby 2.7.2.*



*Send questions to [help@riscv.org](mailto:help@riscv.org).*

### 3.1. Paragraphs

In AsciiDoc, normal paragraphs do not require markup.

To create a new paragraph, put a space after the previous line of text and continue.

#### 3.1.1. Basics of blocks and indents

**If you 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 works only within the context of a numbered or bulleted list.

AsciiDoc/Asciidoctor 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
....
```

This example renders as follows:

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

See [Section 3.8](#) for additional information about blocks.

## 3.2. Headers

When you author 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.

The following example is 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).*

## 3.3. Lists

Unordered lists are created with the **\*** before the list item. Ordered lists require a **.** Add a space

---

between any supporting text at the beginning of a list.

### 3.3.1. Unordered list

To create an unordered list, place a **\*** and a space before an item. Put each new list item on a new line. Add a space between any supporting text at the beginning of a list.

Example:

```
* Priv
* Unpriv
* Debug
```

Example output:

- Priv
- Unpriv
- Debug

### 3.3.2. Ordered list

To create an ordered (numbered) list, place a **.** and a space before an item. Put each new list item on a new line. Add a space between any supporting text at the beginning of a list.

```
. Priv
. Unpriv
. Debug
```

Example output:

1. Priv
2. Unpriv
3. Debug

### 3.3.3. Nested list

To create a nested unordered list, use **\*\*\*** before the nested item.

```
* Priv
** Intro
*** Definitions
** CSRs
* Unpriv
```

Example output:

- Priv
  - Intro
    - Definitions
  - CSRs
- Unpriv

To create a nested ordered list, use `..` before the nested list item.

```
. first item
.. nested item
.. second nested item
. back to original level.
```

Example output:

1. first item
  - a. nested item
  - b. second nested item
2. back to original level.

You can also create an unordered list that contains a nested ordered list (or an ordered list that contains a nested unordered list).

```
* unordered item
.. numbered item
.. second numbered item
* another bullet
```

Example output:

- unordered item
  - a. numbered item
  - b. second numbered item
- another bullet

#### 3.3.4. Add a title to a list

Titles can help introduce your list content.

```
.Ordered list
. Priv
```

- . Unpriv
- . Debug

Example output:

*Ordered list*

1. Priv
2. Unpriv
3. Debug

### 3.4. Hyperlinks and cross references

Asciidoctor automates some linking as follows:

- Recognizes hyperlinks to Web pages and shortens them for readability.
- Automatically creates an anchor for every section and discrete heading.

#### 3.4.1. Hyperlinks

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

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

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

#### 3.4.2. Cross references

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

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

This example renders as:

[Section 4.1](#) describes how index markers work.

For more information about options, see [Cross References](#).

### 3.5. Tables

By using tables, you can group information into logical units, which can make the information presented easier to understand.

### 3.5.1. General rules for tables

Follow these general rules when you create a table.

- Avoid tables in the middle of lists.
- Do not use tables to lay out a page. For example, if you have a long list of items, do not use a 2 column table to save space. The information should make sense side by side.
- Do not create a table that has a single row or a single column, unless you are following a previous layout. For example, if each section in a chapter includes a table of options, then use a table even if one of the sections has only a single option.
- Always use table headers and captions to make your tables more accessible.
- Use introductory sentences for your table. For example, "The following table contains the options for the CSR." You can use either a period or a colon for your introductory sentence.
- Do not refer to the "table above" or the "below table". Use words such as "The following table" or "The preceeding table".



**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.

### 3.5.2. Simple table

The following example shows a simple table with 2 rows and 2 columns. To indicate a new row, put a empty line between them.

```
[cols="1,1"]
|==
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|==
```

Results in the following table.

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

### 3.5.3. Adding table headers

Headers can add additional information to your table, making them easier to understand.

You can add a header row by adding the first row of cells directly in a line.

```
[cols="1,1,1"]
|==
```

```
|Col 1, header row|Col 2, header row|Col 3, header row

|Cell in col 1, row 2
|Cell in col 2, row 2
|Cell in col 3, row 2
|===
```

Or you can use the **header** option.

```
[%header,cols="1,1,1"]
|===
|Col 1, header row
|Col 2, header row
|Col 3, header row

|Cell in col 1, row 2
|Cell in col 2, row 2
|Cell in col 3, row 2
|===
```

Either table renders with table headers.

Col 1, header row	Col 2, header row	Col 3, header row
Cell in col 1, row 2	Cell in col 2, row 2	Cell in col 3, row 2

### 3.5.4. Table captions

The **book\_header.adoc** file in the docs-templates repo sets the **full** cross-reference attribute to enable captions to display from targets in the anchors. This ability allows you to set captions for tables, blocks, and illustrations. If you do not provide a caption, Asciidoctor defaults to the *basic* cross reference style.

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

The following table, <<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 |No<math>[\$^{\{1\}}\$]</math> |No |Yes
```

```
|Software is oblivious |No |No |Yes |Yes $:[\${2}]$ 
|Handled by environment |No |Yes |Yes |Yes
|===
```

The following table, [Table 1](#) shows the characteristics of each kind of trap.

Table 1. Characteristics of traps.

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

3.5.5. AsciiDoc tables from CSV data.

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

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

The previous example 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](https://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

The following example 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.

The previous example results in the following table.

Table 2. proposed 16-bit encodings-1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	instr uction
100			1	0	0	0	field		0	0	00		field	00		mne moni c1
100			1	0	0	field			bit	1	field			00		mne moni c2
110			1	0	0	field			1	0	field			00		mne moni c3
This row spans the whole table																
100			1	1	1	field								00		mne moni c4



### 3.6. Unicode symbols

For PDFs, five-digit unicode symbols generally don't work and some other unicode symbols are buggy. This failure happens because the Ruby asciidoctor-pdf toolchain makes use 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](https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references) that might be useful:

As an example, ♦ is encoded as follows:

```
&#9830;
```

Table 3. Useful unicode for specifications

sym	num	name
^	94	caret
□	136	
⋮	8942	vdots
♦	9830	name
"	0034	name
w	0077	w
∴	8756	therefore
#	9839	sharp
ш	1096	shcy
ϰ	982	piv varpi
ω	969	omega
℘	8472	weierp wp
Σ	8721	sum
∞	8734	infin
∫	8747	integral
≠	8800	not equal to
≤	8804	le
≥	8805	ge
≈	8776	numerical approximation
D	68	mathematical D?
⇒	8658	rightwards double arrow
X	88	Latin Capital x
Χ	967	Greek x
×	215	times
☑	9745	boxed checkmark
r	114	latin small letter r

For many other symbols, use `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

### 3.7. 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 3.6](#)).*

#### 3.7.1. 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:

$1234^{\text{NOTE}}$

For subscripts, use tildes:

```
C~2~ H~6~
```

With the following result:



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 H<sub>2</sub>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.

### 3.7.2. Latexmath

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

```
latexmath:[$C = \alpha + \beta Y^{\gamma} + \epsilon$]
```

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 expressions inline and preferentially make use of unicode or superscripts and subscripts when possible.*

### 3.7.3. 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
++++
```

$$\textit{sqrt}(4) = 2$$

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

For updates to asciidoctor-pdf, see [github.com/asciidoctor/asciidoctor-pdf](https://github.com/asciidoctor/asciidoctor-pdf).

### 3.8. Admonition blocks

Five types of standard [admonition blocks](#) are available in AsciiDoc. RISC-V uses these five types with the default icons.



*The admonition type is not displayed, only the icon.*

#### Note

Highlight extra information that should stand out from the rest of the text. A "by the way, you should know this" statement.



*example of note type*

#### Caution

Cautions users about a condition or action that can lead to bad outcomes such as personal injury or damage equipment. A "we don't recommend" statement.



*example of caution type*

#### Warning

Warns users about a situation that is undesirable. A "Do not do this!" or "You must do this" statement.



*example of warning type*

#### Important

Information that a user must have. For example, "After you set your password, you cannot change it."



*example of important type*

#### Tip

Used for Non-normative text such as clarification or hints for implementers or to convey design rationale and why other options were discontinued. RISC-V tech team is working on a solution that will allow these admonitions to be turned off.



*example of tip type*

As a general rule, follow these guidelines for admonitions:

- Understand that admonitions are interruptions. They should be relevant to the topic, but not necessary. If the reader skips reading it, they can still succeed.
- If the information is necessary, make it part of the topic and even add a heading. Don't put it in an admonition.
- Limit admonitions to a maximum of 3 on a page (standard PDF page - longer HTML pages can use more.)
- Do not include results, steps, or prerequisites in admonitions.
- Make your admonition clear and concise.

### 3.8.1. Single paragraph admonition

For a single paragraph admonition, use a double colon:

**NOTE:** Note content.

which renders as:



*Note content.*

### 3.8.2. Admonition blocks

An admonition block can contain any AsciiDoc content.

**[IMPORTANT]**

====

As a general rule, follow these guidelines for admonitions:

- \* Understand that admonitions are interruptions. They should be relevant to the topic, but not necessary. If the reader skips reading it, they can still succeed.
- \* If the information is necessary, make it part of the topic and even add a heading. Don't put it in an admonition.
- \* Limit admonitions to a maximum of 3 on a page (standard PDF page - longer HTML pages can use more.)
- \* Do not include results, steps, or prerequisites in admonitions.
- \* Make your admonition clear and concise.

====

which renders as:



*As a general rule, follow these guidelines for admonitions:*

- *Understand that admonitions are interruptions. They should be relevant to the topic, but not necessary. If the reader skips reading it, they can still succeed.*
- *If the information is necessary, make it part of the topic and even add a heading. Don't put it in an admonition.*
- *Limit admonitions to a maximum of 3 on a page (standard PDF page - longer HTML pages can use more.)*
- *Do not include results, steps, or prerequisites in admonitions.*
- *Make your admonition clear and concise.*

### 3.8.3. Admonition with a title

You can add a title to your admonition block.

[WARNING]

.Security vulnerability

====

\*Be aware that RLB introduces a security vulnerability if it is set after the boot process is over.\* Use with caution, even when you use it temporarily. Editable PMP rules in M-mode gives a false sense of security since it only takes a few malicious instructions to lift any PMP restrictions this way. It doesn't make sense to have a security control in place and leave it unprotected. Rule Locking Bypass is only meant as a way to optimize the allocation of PMP rules, catch errors durring debugging, and allow the bootrom/firmware to register executable `_Shared-Region_` rules. If developers / vendors have no use for such functionality, they should never set `mseccfg.RLB` and if possible hard-wire it to 0. In any case *RLB should be disabled and locked as soon as possible*.`

====

Rendered:



*Security vulnerability*

Be aware that RLB introduces a security vulnerability if it is set after the boot process is over. Use with caution, even when you use it temporarily. Editable PMP rules in M-mode gives a false sense of security since it only takes a few malicious instructions to lift any PMP restrictions this way. It doesn't make sense to have a security control in place and leave it unprotected. Rule Locking Bypass is only meant as a way to optimize the allocation of PMP rules, catch errors durring debugging, and allow the bootrom/firmware to register executable `Shared-Region` rules. If developers / vendors have no use for such functionality, they should never set `mseccfg.RLB` and if possible hard-wire it to 0. In any case RLB should be disabled and locked as soon as possible.

#### 3.8.4. RISC-V admonition icon colors

The admonition icons are set in `risc-v_spec-pdf.yml`. RISC-V uses custom colors, as indicated in the [Table 5](#).






	<i>note</i>
	<i>tip</i>
	<i>warning</i>
	<i>caution</i>
	<i>important</i>

Table 5. Customized colors for icons

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

### 3.9. 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:

```

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

```

*Listing 1. A spinlock with fences*

### 3.10. 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](https://github.com/riscv/docs-templates)).
- Git features.
- a few procedures associated, specifically, with Git updates.

#### 3.10.1. Indicate changes

Indicators for the changed lines must be inserted manually:

```
[.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#
```

### 3.10.2. Check for changed lines before a **git commit**

You can double check for all changed lines before committing by 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 example:

```
5
38
109
237
```

## 3.11. 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<sup>[1]</sup>
- tertiary
- apocalyptic

A bold statement!<sup>[2]</sup>

Another outrageous statement.<sup>[2]</sup>

### 3.12. 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.*

31	25	24	20	19	15	14	12	11	7	6	0											
0	0	0	0	1	0	0		rs2		rs1				rd		0	1	1	1	0	1	1
ADD.UW								ADD.UW								OP-32						

*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 decision-making 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.*

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

*Listing 2. Source code block in a sidebar*

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

[2] Opinions are my own.

## Chapter 4. Index and bibliography

An index and bibliography are included in the main priv and unpriv docs. You can develop your own index and bibliography for your stand alone document, but if it is to be merged into the main docs, you must merge the index and bibliography as well.

### 4.1. 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)), ①
was to carry Excalibur (((Sword, Broadsword, Excalibur))). ②
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. ③
indexterm:[knight, Knight of the Round Table, Lancelot] ④
```

- ① The double parenthesis form adds a primary index term and includes the term in the generated output.
- ② 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).
- ③ The inline macro `indexterm2\[primary]` is equivalent to the double parenthesis form.
- ④ 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.

## 4.2. Bibliography and references

There are two ways of handling bibliographies:

- By using the automated features provided by `asciidoc-bibtex` (preferred).
- Creating manual entries to which you can create links from the text in the body of your document (deprecated).

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

### 4.2.1. Creating an automated bibliography with `asciidoc-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 `asciidoc-bibtex` to work, install the Ruby gems as documented in the `docs-templates` README file.

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

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

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

When you run `asciidoc-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 following this pattern:

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

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

Add age numbers (locators) following this pattern:

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

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

Add pretext following this 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 prevent problems with other appendices, keep the index as the second-to-last appendix and the bibliography as the last appendix in your 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.*

The following examples are 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}
}

```

#### 4.2.2. Manual bibliography procedures (deprecated)

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>>.



## Chapter 5. Graphics

Graphics help people learn, break up text, and can overall improve your document content.

While AsciiDoc can render graphics in all popular formats, by far the highest quality graphics rendering is from **.svg** format. This format is the preferred graphic type to use in RISC-V documents.

The [asciidoc-diagram extension](#) supports numerous diagram types. Some types that are common are:

- [Section 5.2](#) Wavedrom diagrams
- [Section 5.4](#) Graphviz diagrams
- [Section 5.6](#) Bytefield diagrams
- [Section 5.5](#) Ditaa diagrams

You can certainly use one of the other supported types, but know that they might cause issues with the build. Please contact the RISC-V docs team before using them.

### 5.1. Graphics best practices

Follow these guidelines for graphics.

- Store your graphics in a subfolder. For the RISC-V main ISA doc, this folder is the **images** [folder](#). Place your graphic in the subfolder that corresponds to its type.
- The build process creates the final image object. Do **not** create any generated image files into GitHub.
- Introduce your graphic with a lead-in sentence. "The following image shows ..."
- Use "following" and "preceding" to locate your image. Avoid using "above" or "below" (Doesn't make sense for people that use screen readers.)
- Avoid using text in your image. If it can be included as regular text, then don't include it as part of the image. (screen readers again).
- Images should support your text. Do not put important information in only an image.

### 5.2. Wavedrom diagrams in specifications

Wavedrom diagrams are used mainly for registers. To specify a wavedrom file, create a **json** file and then call it from your text. For more information, see [WaveDrom sequence diagrams](#).

The following json-formatted script, when added within an AsciiDoc block with the macro indicators **[wavedrom, ,svg]**, embeds 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, , ]` includes 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 cases while preserving consistency in how the build renders figure titles, we use a minimalistic, white graphic with the filename `image_placeholder.png` that blends in with the page background. The following example shows 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:

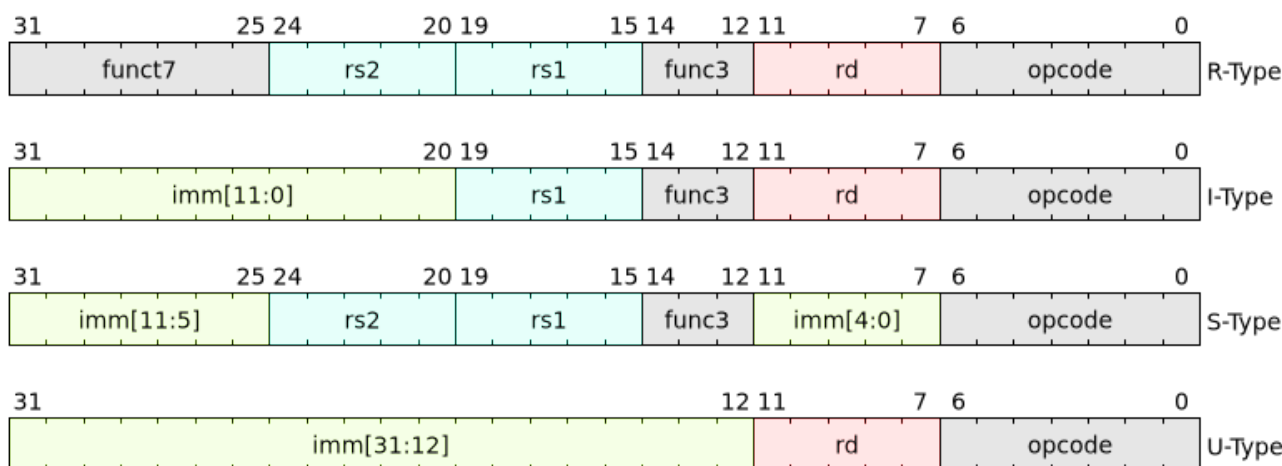


Figure 1. Test for wavedrom

1. Wavedrom code for all diagrams in this illustration are 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, of the figure number and caption.
3. Figure caption.
4. "Invisible" placeholder needed for figure caption to display consistently and correctly.

5.3. Explanation

For the previous example to build into a diagram that includes a figure title, and a figure title and a macro the specifies the diagram type before the code block. You can add a **target** filename and, in addition, specify the image output format to be **svg**.

When prepended to the javascript for a Wavedrom diagram, the following creates **file-name.svg** with the legend **Figure title**:

```
.Figure title
[wavedrom,target="file-name",svg]
```

Following are some examples of Wavedrom diagrams:

```
.Figure title
[wavedrom,target="op-32-add-uw",]
....
{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'] },
]}
....
```

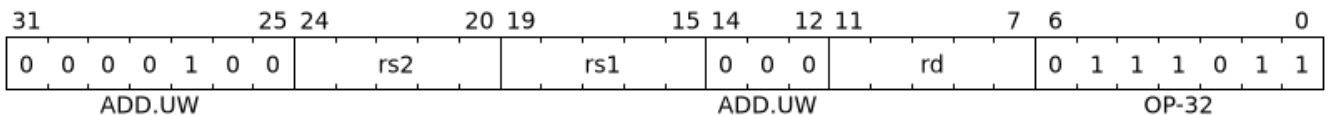


Figure 2. For this example, the output format was not specified, so it defaults to a png.

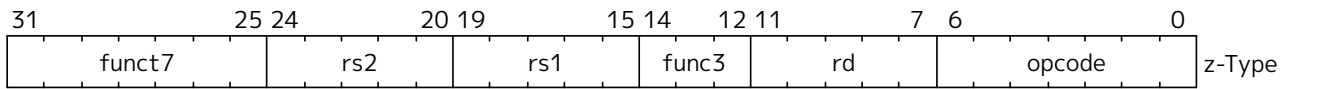


Figure 3. Wavedrom example with svg output specified

Wavedrom Conversion



The following string is lacking macro brackets ([ ]) that should appear after **filename.adoc** because adding the brackets causes 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
```

## 5.4. 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.

### Graphviz digrams

```
.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)
| | $\vdots$  | | $\vdots$ 
|==
```

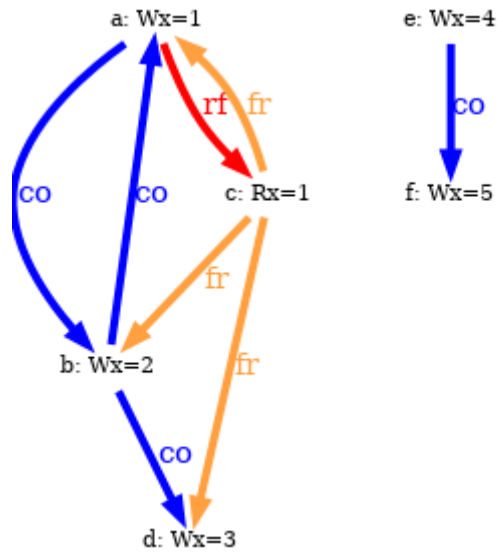


Figure 4. Sample litmus test

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 5.2](#).

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:

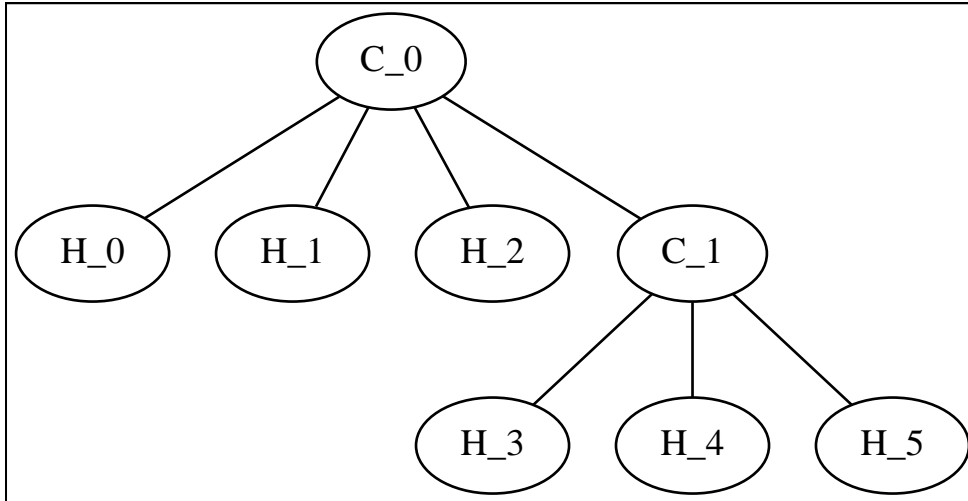


Figure 5. Graphviz s

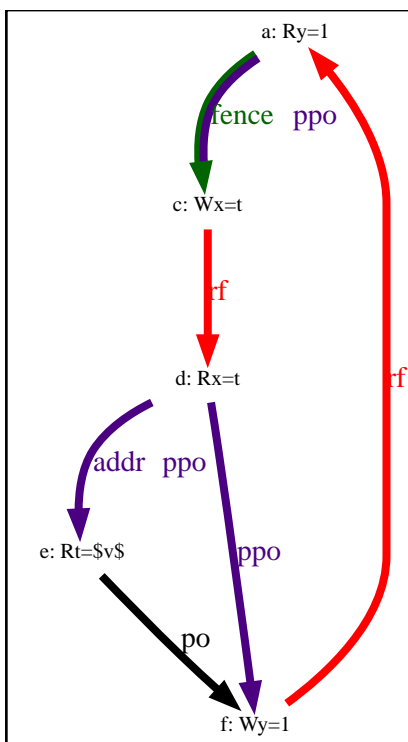


Figure 6. An example graphviz diagram from a specification

## 5.5. Dita diagrams

Following is source for simple dita diagram:

```

[dita,target="image-example",svg]
....

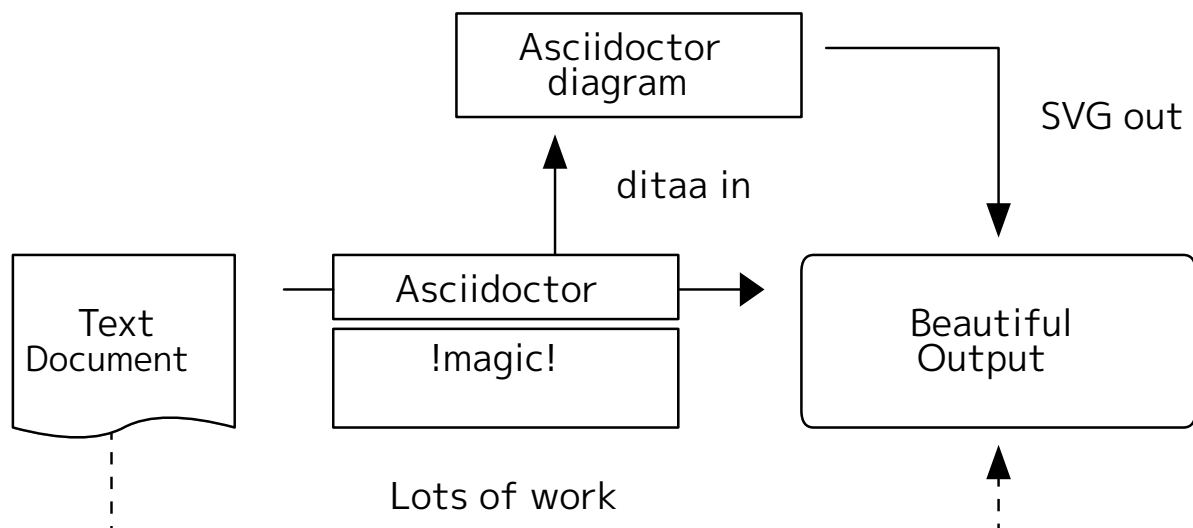
```

```

+-----+
| 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

```





```
:text-anchor "start" :borders {:top :border-unrelated :bottom :border-unrelated}})
(draw-box "0" )
(draw-box "HSXLEN" {:font-size 24 :span 31 :borders {}})
(draw-box "1" {:borders {}})
```

Then you can include it with the following statement (the `{}` are to keep it from rendering):

```
{.Counter-enable (`mcounteren`) register.
include::images/bytefield/examplebyte.adoc[]}
```

After the build, it looks like this example:



Figure 7. Counter-enable (`mcounteren`) register.

## 5.7. Editing Wavedrom diagrams for Unpriv

### 5.7.1. Relevant contextual information

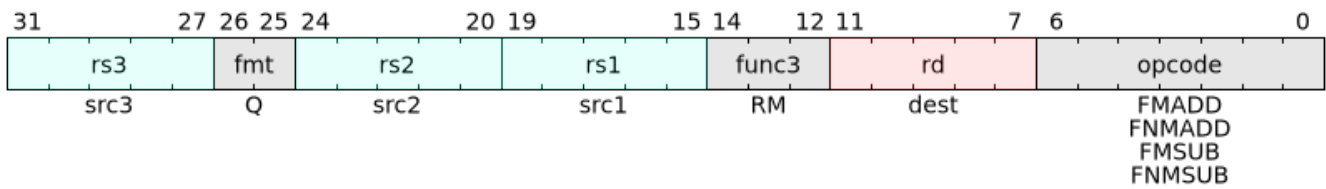
Wavedrom is a utility that is available at [wavedrom.com/](http://wavedrom.com/).

### 5.7.2. 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},
]}
```

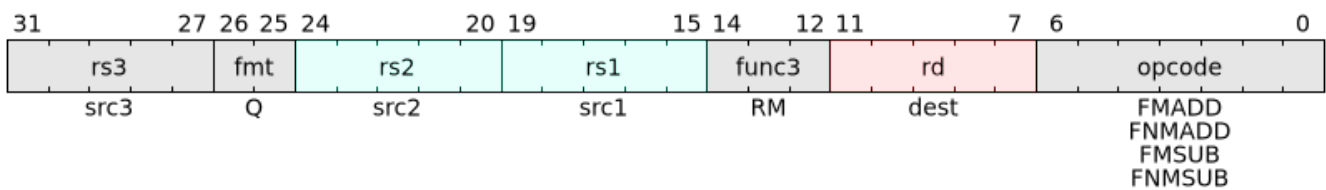
This renders as follows:



```
{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', },
]}
```

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

The output remains the same:



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

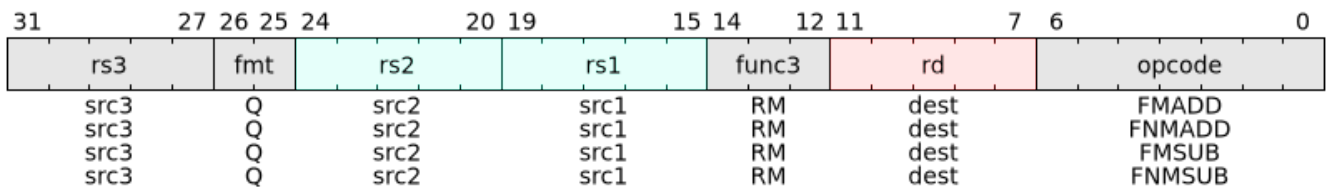
- add **[]** to contain additional values—and--
- follow the convention for commas to contain and separate additional values until all the lines contain the same number of values for **attr** that are in the 'opcodes' row:

```
{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 ```.

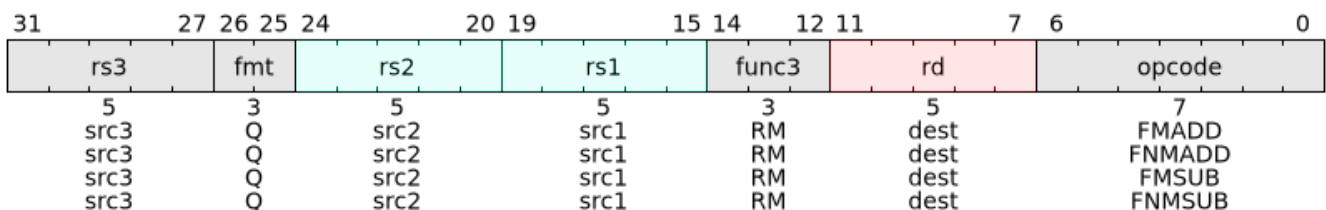
Here's the result:



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.

### 5.7.3. 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](#) ) 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  $\neq$  in Wavedrom diagrams and discovering the above rather confusing results, we decided to use  $\neq$  as a workaround. With the fact that both AsciiDoctor and Wavedrom are evolving, and also the fact that bytefield is being considered as an alternative diagrams rendering solution, it seems possible that this workaround will be temporary.

## Chapter 6. RISC-V Style guidelines

Whether you are creating a new extension or even a stand alone doc for RISC-V, follow these style guidelines to improve readability.

### 6.1. Basic formatting

Follow these basic formatting guidelines.

- Use *italic* for new terms. The *Atomic Layer Deposition* is a layer-by-layer process that results in the deposition of thin films one atomic layer at a time in a highly controlled manner.
- Use **monospace** for the following items:
  - Filenames: **a-st-ext.adoc**
  - Directories: The **src** directory.
  - Paths: [github.com/riscv/riscv-isa-manual/blob/main/src/a-st-ext.adoc](https://github.com/riscv/riscv-isa-manual/blob/main/src/a-st-ext.adoc)
  - Inline code: The **push()** method adds elements to an array.
  - Commands: The **make** command.
- Use **monospace bold** for the following items:
  - Hex numbers. Hex numbers start with **0x**.
  - Binary numbers. Binary numbers start with **0b**.
  - Non-base 10 literals
  - Non-base 10 number ranges

Do not use any special formatting for base 10.

*Formatting guidelines for names*

- Instructions are lowercase (**ld**, **c.lw**)
- Extensions are Capitalized (**A**, **C**, **Zicsr**)
- CSR short names are lowercase (**misa**).
- All of these names are in monospace.

### 6.2. CSR formatting

Use the following guidelines when you document a CSR:

- The acronym CSR must always be capitalized.
- When plural, lowercase the **s**. CSRs.
- The word **Register** is capitalized in title, but lowercase in text. "Supervisor Status (**sstatus**) Register". But "The **sstatus** register..."
- When in doubt, use "CSR" to indicate the type of register. You can then intermingle CSR and register in the text. So "The **misa** CSR is used to..." And then later in the paragraph, you can use "register". As in, "This register also does this other thing."

- Avoid starting a sentence with a CSR name.
- In a title, the format is “Long name (**short name**) Register. All other references in that section can use the short name, but it must be in monospace. So “Supervisor Status (**sstatus**) Register”. The rest of the references to that register can be “the **sstatus** CSR.
- The short name is always lower case and monospace: **sstatus**.
- As a general rule, whenever you use a term in tics, it should be followed by what the thing is to help with translation. So avoid statements similar to this one: “**misa** also helps distinguish different variants of a design.” And instead use this one: “The **misa** CSR also helps distinguish different variants of a design.”
- Fields for registers are formatted in this style: **register**.FIELD. For example, **sstatus**.SPP.

## 6.3. Table formatting

Follow these formatting rules when you create a table.

- Align tables to center with the options **float="center"** and **align="center"**.
- Use table header coding to indicate a header. Don’t use a different font, color, or any background indication.
- Sort rows in either a logical order or by alphabetizing the rows and columns.
- If your table is long or complicated, consider creating multiple tables. Remember that tables can be hard for screen readers to parse. If your table is complex, be sure that the contents are described in the text.
- Use table captions to describe your table contents. Captions appear after the table and are controlled by the theme.
- If you use footnotes in your table, make sure they appear immediately after the table.

### 6.3.1. Column header formatting

Follow these column header rules.

- Use sentence case.
- Write concise headings and omit articles (a, an, the).
- Don’t end with punctuation, including a period, an ellipsis, or a colon.
- Use table headings for the first column and the first row only.

### 6.3.2. Punctuation in tables

Follow these punctuation rules for tables.

- If all cells in a table column are complete sentences, then end each cell with a period.
- If all cells in a table column are sentence fragments, then do not use a period to end each cell.
- If cells in a table column contain a mixture of complete and fragmented sentences, then first try to make them all parallel - either all complete sentences or all sentence fragments. However, if this approach is impractical, then punctuate each cell independently, and punctuate appropriate for

that individual cell.

- Do not end column headers with punctuation, including a period, an ellipsis, or a colon.

## Chapter 7. Best practices

This section contains suggested best practices for clear, concise, and consistent content.

### 7.1. Use present tense

Yes	No
Cache-management operation instructions perform operations on copies of data in the memory hierarchy.	Cache-management operation instructions will perform operations on copies of data in the memory hierarchy.

Exception: Use future or past tense if it is required to convey the correct meaning.

### 7.2. Use active voice

Yes	No
You can use the RVWMO memory model	The RVWMO memory model can be used
The RVWMO memory model enables architects	Architects are enabled by the RVWMO memory model.

Exception: Use passive voice if active voice leads to an awkward construction.

### 7.3. Use simple and direct language

Use simple and direct language. Avoid using unnecessary phrases, such as saying "please." Direct language is easier to translate.

Yes	No
To build a chip,	In order to build a chip,
See the Hypervisor extension.	Please see the Hypervisor extension.
View the register.	With this next command, we'll view the register.

### 7.4. Address the reader as "you"

Using "we" in a sentence can be confusing, because the reader might not know whether they're part of the "we" that you're describing. Does it mean the RISC-V team, the RISC-V members, open source people, hardware people, or even everyone?

Yes	No
You can use the <b>misa</b> CSR	We'll use the <b>misa</b> CSR
In the preceding output, you can see	In the preceding output, we can see

An exception to this rule is the rationale sections.

### 7.5. Avoid Latin phrases

Prefer English terms over Latin abbreviations. Latin terms can be difficult for translation because it adds an additional language to translate.



Yes	No
For example,	e.g.,
That is,	i.e.,

## 7.6. Avoid jargon and idioms

Some readers speak English as a second language. Avoid jargon and idioms to help them understand better.

Yes	No
Internally,	Under the hood,
Stop trying.	Chutar o pau-da-barraca (which translates to "kicking away the tent pole")

## 7.7. Avoid statements about the future

Avoid making promises or giving hints about the future. If you need to talk about an alpha feature, put the text under a heading that identifies it as alpha information.

An exception to this rule is documentation about announced deprecations targeting removal in future versions.

## 7.8. Avoid statements that will soon be out of date

Avoid words like "currently" and "new." A feature that is new today might not be considered new in a few months.

Do	Don't
In version 1.4,	In the current version,
The pointer masking extension provides	The new pointer masking extension provides

## 7.9. Avoid words that assume a specific level of understanding

Avoid words such as "just", "simply", "easy", "easily", or "simple". These words do not add value and can actually make a user feel not up to the task.

Do	Don't
Include one command in	Include just one command in
Run the command	Simply run the command
You can remove	You can easily remove
These steps	These simple steps



*These guidelines were adapted from the [Documentation style guidelines](#) for Kubernetes in August of 2024.*

## 7.10. Other style guidelines

Other style guidelines for reference:

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

## Chapter 8. Word usage

### Above

Avoid using directional words. Above and below do not translate well to screen readers. Instead, use "previous" or "following".

### Acronyms

Acronyms and other shortened forms of words must spell out the acronym at first use in a section. Also, you can add them to the [\[github.com/riscv/riscv-glossary/blob/main/src/glossary.adoc\]](https://github.com/riscv/riscv-glossary/blob/main/src/glossary.adoc)glossary for a handy reference.

### After (once)

Use "after" to indicate a sequence of events. Use "once" to indicate "one time".

### Also

use to mean additionally rather than alternatively.

### As

Don't use "as" to mean "because". For example, don't say "Use the correct version as the wrong version can cause issues." Instead, "Use the correct version because the wrong version can cause issues." Use "as" to compare.

### Because (since or as)

Use "because" to mean "for the reason that" or "due to the fact that". "As" is a comparison. "Since" means a timeline.

### Before (versus "prior," "previous," or "preceding")

- If possible, replace "prior to" with "before" as "before" is a little less formal.
- Use "previous" to indicate something that occurred at an unspecified time earlier.
- Use "preceding" to indicate something that occurred immediately before.

### Below

Avoid using directional words. Above and below do not translate well to screen readers. Instead, use "previous" or "following".

### Can (might, must, may, should, shall, will)

- Use "can" to indicate capability: "This option can cause your system to fail."
- Use "might" to indicate possibility: "This option might affect your system performance."
- Don't use "may," which is ambiguous because it could mean either capability or permission.
- Use "should" to indicate a recommended, but optional action. Consider using an alternative phrase instead, such as "we recommend." Do not use "should" to indicate something that might happen. "After you push the power button, the system should turn on." Instead, be bold! "After you push the power button, the system turns on."
- Use "must" to indicate a required action or condition. "The system must be powered on."
- Use "shall" to indicate something must happen, but has not yet occurred. "The state of the **BUSY** bit shall change only in response to a write to the register."
- Use "will" very sparingly. Use the present tense for most technical documentation. Use future or past tense if it is required to convey the correct meaning only.

## Contractions

Use common contractions as they set a conversational tone. For example, it's, isn't, can't, don't, and so on.

## Following

Don't use "following" by itself. Don't say "See the following". Instead, use "See the following list".

## If (whether)

Use "if" as a condition, such as logic. "If a, then b." Use "whether" to indicate choice or alternative. "Event a happens, whether event b does or not".

## Latin phrases

Avoid abbreviations such as etc., e.g., i.e. They do not translate well. Instead use "and so on" and "for example,". If you do use e.g. or i.e., then know that e.g. means "for example" and i.e means "in other words". With e.g., it is understood that there are more examples than just the ones listed; "The colors of the rainbow, e.g. red, yellow, and green". With i.e., however, it is intended as a replacement for the previous text. "The primary colors, i.e. red, blue, and yellow".

## Legal

Use only to indicate that something is allowed because of a law. "RISC-V processors are legally available." Avoid using when something is allowed. Instead, use "valid".

## Left (and right)

Avoid using these words if at all possible. If you do use them, use "left" and "right" and not "lefthand" or "righthand".

## Like

Use "like" to compare or draw similarities. To provide examples, use "such as".

## Might (can, must, may, should, shall, will)

- Use "can" to indicate capability: "This option can cause your system to fail."
- Use "might" to indicate possibility: "This option might affect your system performance."
- Don't use "may," which is ambiguous because it could mean either capability or permission.
- Use "should" to indicate a recommended, but optional action. Consider using an alternative phrase instead, such as "we recommend." Do not use "should" to indicate something that might happen. "After you push the power button, the system should turn on." Instead, be bold! "After you push the power button, the system turns on."
- Use "must" to indicate a required action or condition. "The system must be powered on."
- Use "shall" to indicate something must happen, but has not yet occurred. "The state of the **BUSY** bit shall change only in response to a write to the register."
- Use "will" very sparingly. Use the present tense for most technical documentation. "The system will power on" becomes "The system powers on." Use future or past tense if it is required to convey the correct meaning only.

## Numbers

Use Arabic for numbers greater than 10. Use words for numbers 1 through 10. Except in the following cases.

- If the numbers are values, use Arabic. "Valid input is 1-10."

- If there is a mix of numbers less and greater than 10, use Arabic. "2, 3, 5, 7, 11, 13, 17, and 19 are the prime numbers between 1 and 20."
- Use the word if the number begins a sentence. "One is the loneliest number." If the number is a value, rewrite the sentence so that the number doesn't begin the sentence. "2 is the correct answer" can be rewritten to be "The correct answer is 2."

### Once (after)

Use "after" to indicate a sequence of events. Use "once" to indicate "one time".

### Prior (versus "before," "previous," or "preceding")

- If possible, replace "prior to" with "before" as "before" is a little less formal.
- Use "previous" to indicate something that occurred at an unspecified time earlier.
- Use "preceding" to indicate something that occurred immediately before.

### Re- words

In general, words with the prefix **re** can be written as one word without a hyphen. The only exception is **re-create**, meaning to create again.

### Should (can, might, must, may, should, shall, will)

- Use "can" to indicate capability: "This option can cause your system to fail."
- Use "might" to indicate possibility: "This option might affect your system performance."
- Don't use "may," which is ambiguous because it could mean either capability or permission.
- Use "should" to indicate a recommended, but optional action. Consider using an alternative phrase instead, such as "we recommend." Do not use "should" to indicate something that might happen. "After you push the power button, the system should turn on." Instead, be bold! "After you push the power button, the system turns on."
- Use "must" to indicate a required action or condition. "The system must be powered on."
- Use "shall" to indicate something must happen, but has not yet occurred. "The state of the **BUSY** bit shall change only in response to a write to the register."
- Use "will" very sparingly. Use the present tense for most technical documentation. Use future or past tense if it is required to convey the correct meaning only.

### Since

Use "since" when time is involved. "Since the invention of sliced bread, toasters became popular." Do not use it when you mean "Because".

### That, which, who

- Use "that" (without a trailing comma) to introduce clauses that are required for the sentence to make sense.
- Use "that" even if the sentence is clear in English without it: "Verify that the computer is off."
- Use "which" (with a trailing comma) to introduce clauses that add supporting information but are not required for the sentence to make sense.
- Use "who" to introduce clauses referring to people.

### This, those, these

Provide a noun after words such as this, those, and these. For example, "This is the output of the command." Instead use "This example is the output of the command."

## Time frame

Write as 2 words, no hyphen.

## Using

Try not to use "using" by itself. Replace with "by using" or "with". "Using" can be either a noun or a participle, which can causing translation issues. You can use "Using" at the beginning of a sentence such as "Using RISC-V standards to design your chip".

## Whether (if)

Use "if" as a condition, such as logic. "If a, then b." Use "whether" to indicate choice or alternative. "Event a happens, whether event b does or not".

## Will (can, might, must, may, should, shall, will)

- Use "can" to indicate capability: "This option can cause your system to fail."
- Use "might" to indicate possibility: "This option might affect your system performance."
- Don't use "may," which is ambiguous because it could mean either capability or permission.
- Use "should" to indicate a recommended, but optional action. Consider using an alternative phrase instead, such as "we recommend." Do not use "should" to indicate something that might happen. "After you push the power button, the system should turn on." Instead, be bold! "After you push the power button, the system turns on."
- Use "must" to indicate a required action or condition. "The system must be powered on."
- Use "shall" to indicate something must happen, but has not yet occurred. "The state of the **BUSY** bit shall change only in response to a write to the register."
- Use "will" very sparingly. Use the present tense for most technical documentation. Use future or past tense if it is required to convey the correct meaning only.

## Chapter 9. Linting

The linting of code/documentation can be categorized as dealing with formatting, syntax, or semantics. Once the rules for a given project are defined, many of these rules can be checked automatically with tooling.

### 9.1. Tooling

[pre-commit tool](#) is a very useful, open source, well maintained, and popular tool/framework to help automate linting checks in general. It provides some of the following key features:

- [a great number of support hooks for linting](#)
  - including direct support by many of the maintainers of best in class linters (ex: [python linter black](#))
  - [hook plugins](#) allow adding these and versioning to control updates
- support for [creating new custom hooks easily](#)
- can be run before **git commit** (i.e. "pre-commit") so authors can detect and fix problems sooner in the workflow
- can also be run in [continuous integration \(CI\)](#) to ensure checks are run for every pull request

---

## Index

### K

knight

Arthur, King, [26](#), [26](#)



## 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.