Learn RISC-V CPU Implementation and **BSV**

(**BSV**: a High-Level Hardware Design Language)

Rishiyur S. Nikhil

L14: RISC-V: The Drum CPU, using Rules

bluespec

Reminders

Please git clone: `https://github.com/rsnikhil/Learn_Bluespec_and_RISCV_Design`
(git pull for latest version). Repsitory structure:

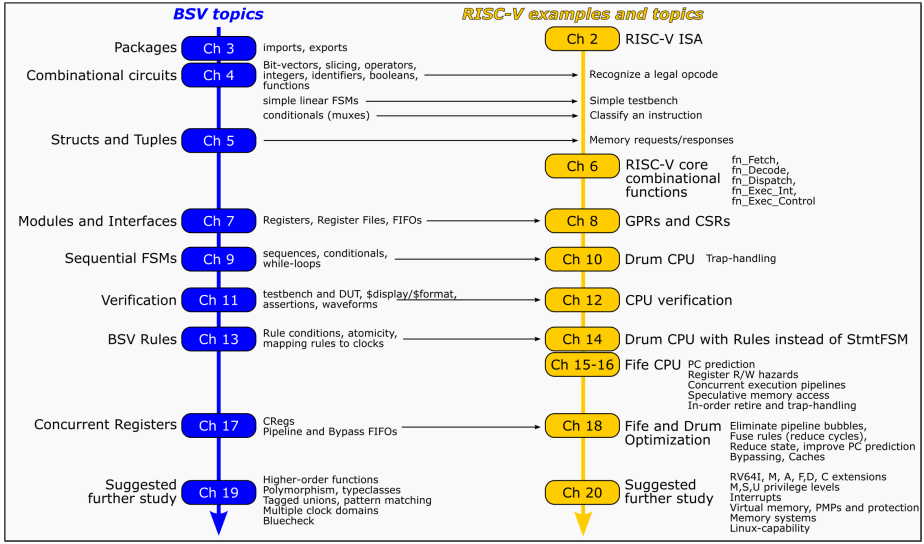
```
./Book_BLang_RISCV.pdf
  Slides/
      Slides_01_Intro.pdf
      Slides_02_ISA.pdf
      ...
  Exercises/
      Ex-03-A-Hello-World/
      Ex-03-B-Top-and-DUT/
      ...
  Code/
      src_Top/
      src_Drum/
      src_Fife/
      src_Common/
      ...
  Doc/Installing_bsc_Verilator_etc.{adoc,html}
```

- Slides and Exercise are numbered in sync with book Chapter numbers.
- For Exercises, please see Appendix E of the book. Some (not all) exercises have associated code in the `Exercises/` directory.

To compile and run the code for exercises, Drum and Fife, please make sure you have installed:

- *bsc* compiler (see `https://github.com/B-Lang-org/bsc`)
- Verilator compiler (see `https://www.verilator.org/`)

Chapter Roadmap



Flow of information between stages in Drum and Fife

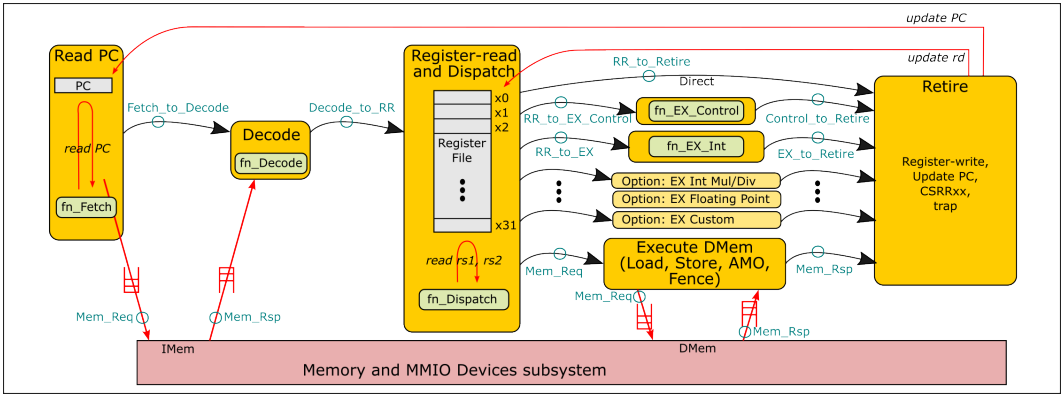


Table of Contents

RISC-V: The Drum CPU, using Rules instead of `StmtFSM`

The Drum CPU, using Rules instead of StmtFSM

This is a short chapter demonstrating a manual translation of the Drum CPU module from one using StmtFSM to one using Rules instead.

This reinforces our claim that StmtFSM does not add any new semantics to **BSV**; it is simply a higher-level notation that can be used fruitfully in certain circumstances ("structured processes").

Please examine the source files in src_Drum/ directory:

CPU.bsv Drum_FSM.bsv Drum_Rules.bsv

CPU.bsv is the common file; it "includes" either Drum_FSM.bsv (StmtFSM version) or Drum_Rules.bsv (Rules version). By examining the latter two files side by side, we can observe the equivalence. Briefly:

- We define an "enum" type that gives a symbolic name to each action-step in the FSM, and introduce a "sequencer" register to hold a value of this type.
- We convert each action in the FSM to a rule
  - whose rule-condition allows it to execute only when the sequencer says it is its turn to execute, and
  - whose rule-body "increments" the sequencer to the next FSM step.

Advantages of the Rules version

`StmtFSM` only permits *structured* process composition, *i.e.,* properly nested sequencing, if-then-else, while-loops.

The Rules version makes it easy to depart from such properly nested structure and "short-circuit" certain flows for better performance.

For example:

- If we detect an exception in rule `rl_Decode` (and, indeed, in any of the other rules), we can immediatelly handle the exception and "jump" back to the Fetch step, thereby saving several cycles.
- In `rl_EX_Control` and `rl_EX_Int`, if there is no exception, we can immmediately perform any register write and "jump" back to the Fetch rule.

End