A DSL for Visualizing Pipelines: A RISC-V Case Study

Lucas Klemmer, Daniel Große

lucas.klemmer@jku.at, daniel.grosse@jku.at
Institute for Complex Systems, Johannes Kepler University Linz, Austria

Abstract

The number of available RISC-V cores is growing rapidly and the openness of RISC-V enabled even smaller teams to develop their own cores. However, the wide variety of RISC-V cores available today and the high modularity of the ISA makes it hard for development tools to keep up with the speed of development, as well as to provide first class support for this growing ecosystem. In this paper, we present a Domain Specific Language *(DSL) for defining processor pipelines. With just a few lines of code in this DSL, information about RISC-V pipelines can be collected from simulation waveforms for further processing. As one application of this DSL, we present a web application that functions as a pipeline rendering backend, helpful for debugging and design understanding.*

Introduction

For a few years now, RISC-V development boards in the embedded category are available, new open-source cores are published regularly, and work is underway to spread RISC-V into more and more domains [1] [2]. Recently, even the first low-cost RISC-V development boards capable of desktop environments have arrived [3] bringing with them the potential to popularize RISC-V with even more developers. Yet, silicon alone is not enough to establish an ISA in the long run, but a large and active software ecosystem as well as first-class development tools are required.

On one hand, these advancements into wildly different domains are a manifestation of RISC-V's modularity and customization potential. On the other hand, it also means that, for RISC-V even more than for other ISAs, no two cores are alike, and thus it is increasingly complicated to provide excellent tool support for as many cores as possible. It also means that the tools must be as flexible as RISC-V itself.

In this paper, we present a DSL for specifying processor pipelines that can be used to extract information about the pipeline from simulation waveforms. We present a web application that can visualize RISC-V pipelines with only a simulation waveform and a few lines of declarations. Our DSL is based on the *Waveform Analysis Language* (WAL) [4, 5], a language for analyzing waveforms, and thus complex pipelines can be easily declared, and the analysis can be enriched by fully fledged programs for further performing complex analyzes on the waveform.

Defining Pipelines

In this section, we present an exemplary pipeline definition for a slightly enhanced version of the pipelined processor introduced in [6]. This processor has 5 pipeline stages consisting of *fetch*, *decode*, *execute*, *memory*, and *writeback* stages, and it implements forwarding using a hazard unit. Fig. 1 shows a block diagram of the processor.

In our DSL, new stages are defined using the `stage` keyword. The behavior of each stage can be configured using a set of defined arguments. These arguments control, for example, when a new instruction enters a stage, when the stage is flushed, or they can compute additional data that should be logged for the stage. The concrete possible arguments are:

value The next value of the pipeline. If unspecified, the value of the previous stage gets sampled. Required for the first stage.

update Controls when the stage gets updated. If unspecified, defaults to true.

stall Controls when the stage is stalled. If unspecified, defaults to false.

flush Controls when the stage is flushed. If unspecified, defaults to false.

log Additional data that gets logged for this stage. Two expressions are required. The first expression is the name under which the data is logged, and the second is the WAL expression that is logged. The expressions can be arbitrarily complex WAL programs.

Listing 1 shows the pipeline definition for Fig. 1. Take, for example, the *decode* stage. This stages loads the next instruction when it is not stalled and when it is not flushed. In this case, the instruction from the *fetch* stage is loaded into the *decode* stage.

The definition of a pipeline is not linked to rendering it. By uncoupling data collection and presentation, both parts are now ready to be reused. Different rendering "backends" can be selected, for example, text output or the web application we present in the following section.

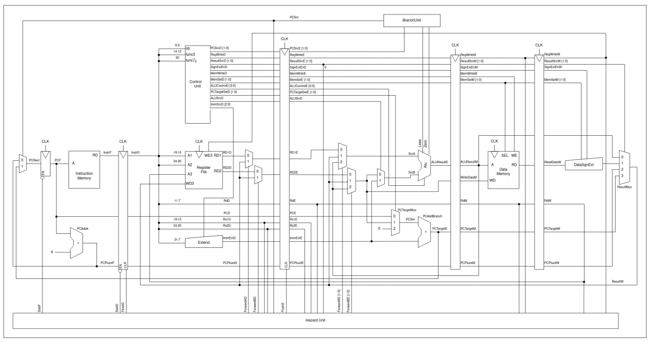


Figure 1: *Enhanced Pipelined Architecture from [6]*

Listing 1: *Pipeline Definition for processor from Fig. 1*

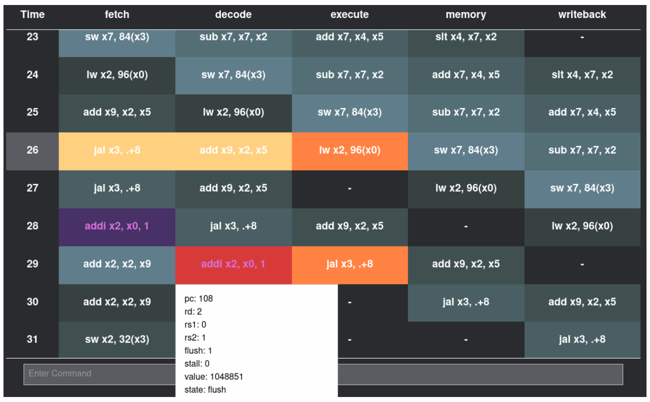
```
(require pipeline)

(stage fetch
       (value tb.dut.dp.instrf@1)
       (stall tb.dut.dp.stallf)
       (log stallf tb.dut.dp.stallf)
       (log pc tb.dut.dp.pcf))

(stage decode
       (update (! tb.dut.dp.stalld))
       (stall tb.dut.dp.stalld)
       (flush tb.dut.dp.flushd)

       (log pc fetch-pc@-1)
       (log rd tb.dut.dp.rdd)
       (log rs1 tb.dut.dp.rs1d)
       (log rs2 tb.dut.dp.rs2d))


(stage execute
       (update (! tb.dut.dp.flushe))
       (flush tb.dut.dp.flushe)
       (log pc decode-pc@-1))

(stage memory)

(stage writeback)
```

Rendering Pipelines on the Web

In this section, we present a web application that shows interactive visualizations of processor pipelines based on pipeline specifications in the previously presented DSL. The application is processor-independent and the visualization depends solely on the pipeline specification made in our DSL.

Modern web technology allows highly dynamic and interactive applications, which is perfect for visualizing complex data such as processor pipelines. Additionally, it makes sharing and cooperation much simpler and allows working on mobile devices as well. After the pipeline was specified, a waveform can be processed to collect the information about all stages during the complete simulation (or parts of it).

Fig. 2 shows our web application rendering the pipeline from Fig. 1 for a given waveform. Each column represents a pipeline stage, and the rows represent the time advancing with each clock cycle. Instructions are shaded differently, to aid the eye while following an instruction through the pipeline. Stalls and flushes are marked in yellow and orange colors respectively, while a single instruction can be specifically highlighted in purple by hovering over it. The visualization can be navigated using the arrow keys and by entering special commands into an input field. Additionally, the pipeline can be searched to highlight cells depending on their contents (e.g., all stages with *add* instructions or all stages that operate on register x5).



Time	fetch	decode	execute	memory	writeback
23	sw x7, 84(x3)	sub x7, x7, x2	add x7, x4, x5	slt x4, x7, x2	-
24	lw x2, 96(x0)	sw x7, 84(x3)	sub x7, x7, x2	add x7, x4, x5	slt x4, x7, x2
25	add x9, x2, x5	lw x2, 96(x0)	sw x7, 84(x3)	sub x7, x7, x2	add x7, x4, x5
26	jal x3, .+8	add x9, x2, x5	lw x2, 96(x0)	sw x7, 84(x3)	sub x7, x7, x2
27	jal x3, .+8	add x9, x2, x5	-	lw x2, 96(x0)	sw x7, 84(x3)
28	addi x2, x0, 1	jal x3, .+8	add x9, x2, x5	-	lw x2, 96(x0)
29	add x2, x2, x9	addi x2, x0, 1	jal x3, .+8	add x9, x2, x5	-
30	add x2, x2, x9		-	jal x3, .+8	add x9, x2, x5
31	sw x2, 32(x3)		-	-	jal x3, .+8

pc: 108
rd: 2
rs1: 0
rs2: 1
flush: 1
stall: 0
value: 1048851
state: flush

Enter Command

Figure 2: *Pipeline rendered as a website*

Conclusions

In this paper, we presented a DSL for visualizing processor pipelines. We have shown how pipelines can be defined in our DSL and how they can be presented, for example as an interactive website.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] A. Walsemann et al. "STRV — a radiation hard RISC-V microprocessor for high-energy physics applications". In: *Journal of Instrumentation* 18.02 (Feb. 2023), p. C02032.

[2] Federico Ficarelli et al. "Meet Monte Cimone: Exploring RISC-V High Performance Compute Clusters". In: *Computing Frontiers*. 2022, pp. 207–208.

[3] *Vision Five 2*. `https://www.starfivetech.com/en/site/boards`.

[4] Lucas Klemmer and Daniel Große. "WAL: A Novel Waveform Analysis Language for Advanced Design Understanding and Debugging". In: *ASP Design Automation Conf.* 2022, pp. 358–364.

[5] Lucas Klemmer and Daniel Große. "Waveform-based performance analysis of RISC-V processors: late breaking results". In: *Design Automation Conf.* 2022, pp. 1404–1405.

[6] David Harris Sarah Harris. *Digital Design and Computer Architecture, RISC-V Edition*. Elsevier, 2021.