Project Report of RISC-V CPU

(Course Project of Computer Architecture)

Weixin Deng (邓伟信)

ACM Class, Shanghai Jiao Tong University

1 Introduction

This project is a RISC-V CPU with a five-stage pipeline, implemented in Verilog HDL. The GitHub repository of my CPU project is: https://github.com/dengwxn/RISC-V-CPU.

2 Implementation

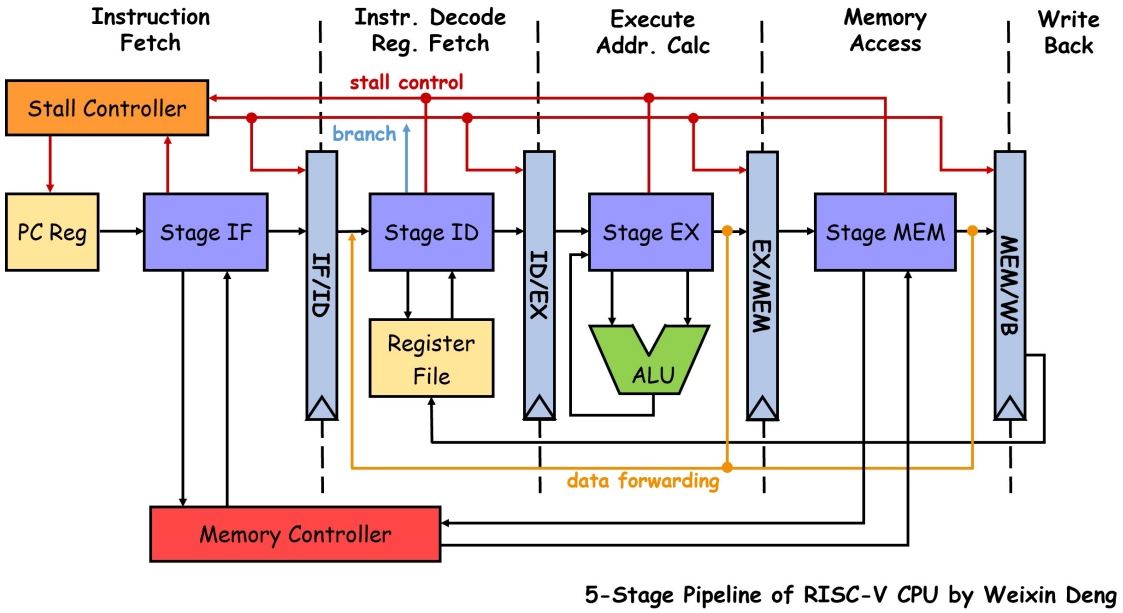

Fig. 1: Overview of CPU Design

My CPU has a RV32I[1] ISA, and has a standard 5-stage pipeline with data forwarding. It always predicts branches are not taken and does not have an instruction cache or a data cache. All test programs run correctly on FPGA.

[1] except *FENCE*, ECALL, EBREAK, CSRR**.

- Data produced by EX and MEM is passed to ID, which can avoid most of RAW data hazards. The pipeline still needs to stall when the producer is a *load* instruction.
- For a *branch* instruction, the branch address is given in ID.
- In this project, the memory module given can read or write only one-byte data per time, so it takes multiple cycles to finish IF and MEM and the pipeline need to stall as well. Then it's necessary to implement an instruction cache and a data cache for better performance but I did not succeed before the deadline.
- Because IF and MEM might need to access memory at the same time if having cache misses, I implement a memory controller which receives requests, sends back data and runs like a deterministic finite automaton.

3 Discussions

Debug I think Verilog HDL is different from other languages like C++ when debugging, though they are similar in syntaxes. One practical way to debug is checking the waveform configuration (WCFG), which is in some sense similar to debug C++ using GDB. Sometimes we'd like to know in *if-else* and *case* blocks which branch my program jumps into. And we can set up some regs which change their values inside different branches and add them in the WCFG (proposed by Zhaoyu Li).

Verilog and FPGA At the beginning, I was not sure about the distinctions between wire and reg, until I found an document.[4] Besides, some potential faults might be invisible in simulations. One common mistake is generating latches, which probably brings unexpected results. And it is thoroughly discussed in this document.[5] I strongly recommend to read these two documents before starting the CPU project, which is helpful to get the CPU running correctly on FPGA.

Pipeline It took me a while to understand functions of combinational logic and sequential logic. In a pipeline, sequential logic drives an instruction to the next stage and latency of combinational logic determines the clock rate and therefore the performance. Technically, my CPU does not have a real pipeline considering that I did not implement an instruction cache. Thus, when I finish fetching the next instruction, all stages of the last instruction are probably finished (unless the last instruction is *load* or *store*). Namely, the correctness of my CPU pipeline remains to be verified.

4 Acknowledgements

I am grateful to Linqi Chen (陈林淇), Qidong Su (苏起东) and Zhaoyu Li (李照宇) who have shared their experience with me and provided much guidance. Besides, I would like to express my gratitude to TA Yunfeng Lin (林耘丰) and TA Yuxi Liu (刘予希) for their great work on the project and the course. And I also thank many other classmates for their direct and indirect help to me.

5 Appendix

References

[1] 雷思磊. 自己动手写 *CPU*, 电子工业出版社, 2014.

[2] Randal E. Bryant and David R. O'Hallaron. *Computer Systems: A Programmer's Perspective, (CS:APP)*, Third Edition, 2015.

[3] Zhou Fan's RISC-V CPU project. https://github.com/Evensgn/RISC-V-CPU.

[4] Chris Fletcher. *Verilog: wire VS. reg*. https://inst.eecs.berkeley.edu/~cs150/Documents/Nets.pdf.

[5] Chris Fletcher. *Verilog: always @ Blocks*. http://inst.eecs.berkeley.edu/~cs150/fa08/Documents/Always.pdf.