Project Report of RISC-V CPU

(Course Project of Computer Architecture)

Zhou Fan (范舟)
ACM Class, Shanghai Jiao Tong University

1 Introduction

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

2 Design

2.1 Features

Main features of this RISC-V CPU are briefly introduced in the table below.

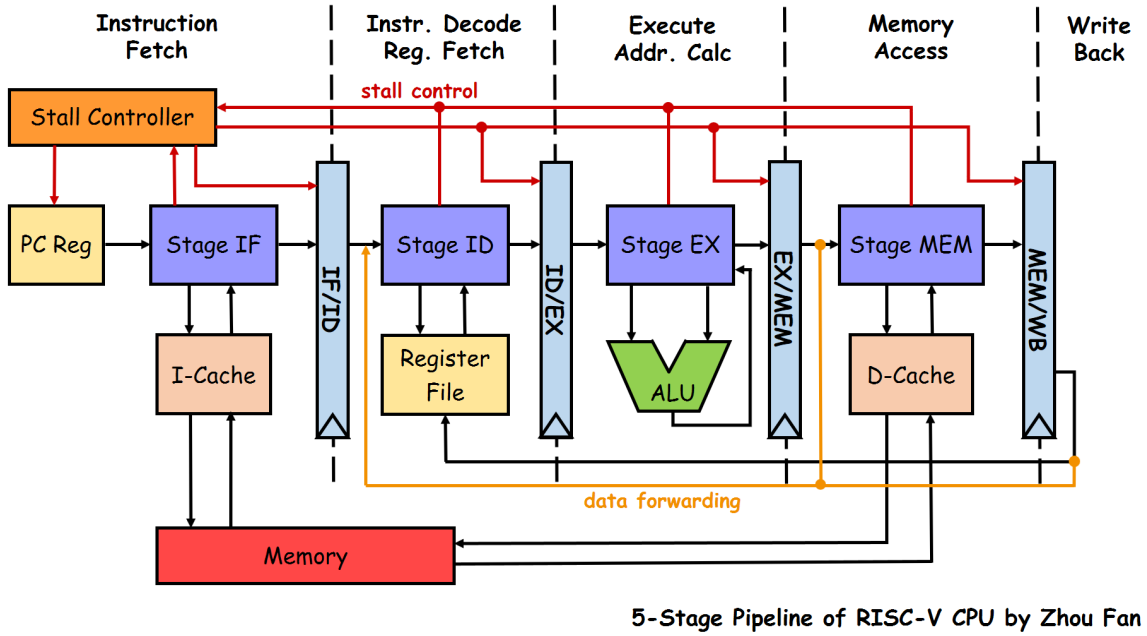
Feature	RISC-V CPU
ISA	RISC-V (RV32I subset)
Pipelining	5 stages
Data forwarding	complete forwarding path
Cache	N-way set associate I-cache and D-cache[1]
UART module	passed simulation [2]
Security	perfect proof against Meltdown and Spectre attack [3]

1. The cache is based on Zhekai Zhang's (张哲恺) code[4].
2. UART module has not passed test on FPGA yet for the limited time. I re-designed part of CPU code to avoid hidden danger on FPGA, and it may need some more debugging.
3. Just kidding ;-) That's because the CPU is not with branch prediction or out-of-order execution.

2.2 Specification

The CPU has a standard 5-stage pipeline with complete path data forwarding. Data produced by EX stage or MEM stage is passed to ID stage, which avoids most of RAW data hazard under ideal conditions (causes stall only if producer is a load instruction).

The picture below shows structure of the CPU design, each module is implemented in a single verilog file. Red paths show the stall control flow, while orange ones show data forwarding path.



- Instruction fetching, Load/Store instruction or data dependency may cause stall of pipeline, the logic of stall is managed by a stall controller module. It receives stall requests from IF/ID/MEM stage, and emit stall signals to modules that should pause.
- For program test on FPGA without capable memory, the CPU uses UART protocol to communicate with PC, where runs a memory simulator written in C++.
- Latency of UART communication makes it significant to use a cache, the cache is N-way associate using LRU replacement policy. It is based on code from Zhekai Zhang's MIPS CPU project.

The CPU with cache and UART module passed simulation in Xilinx Vivado using multiple test data. But something goes wrong when tested on FPGA (Basys 3), which is explained in the next section.

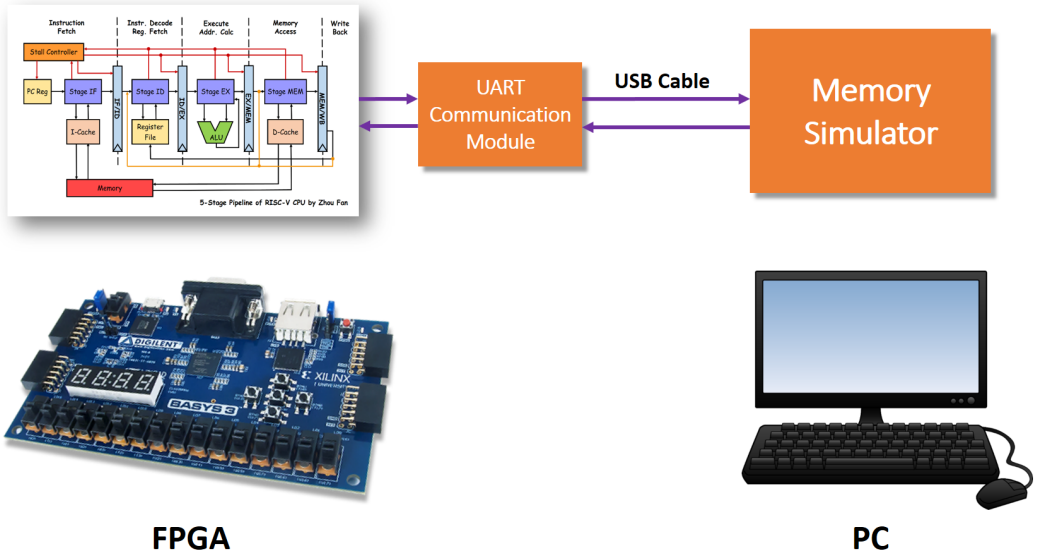


Fig. 1: Communication with memory simulator using UART protocol

3 Puzzles & Thinkings

Efficiency Time efficiency is always a significant issue in CPU design. When I wrote code for stall logic and Branch/Jump instruction processing part, I tried to minimize number of unused clock cycle. But when UART latency is considered, all of those optimization lose their meaning: UART communication is the only bottleneck of the pipeline, and even the pipeline is not a pipeline any more, for an instruction can easily goes through all stages before the next instruction is fetched.

4 Acknowledgements

Special thanks would go to Zhanghao Wu (吴章昊) for his instructive discusstions and useful suggestions on this project. I would like to express my gratitude to TA Zhekai Zhang (张哲恺) as well, for his MIPS CPU project (especially the code of cache and UART module) and much work for this assignment. I am also indebted to many other classmates for their direct and indirect help to me.

5 Appendix

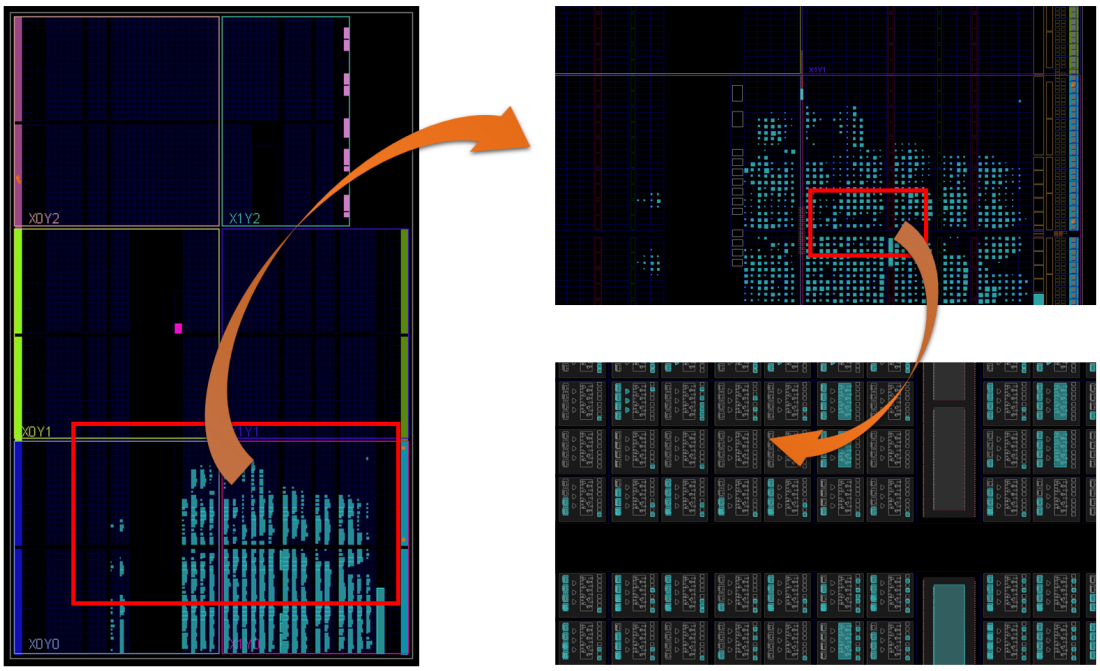


Fig. 2: Implementation on Basys 3 FPGA, using Xilinx Vivado

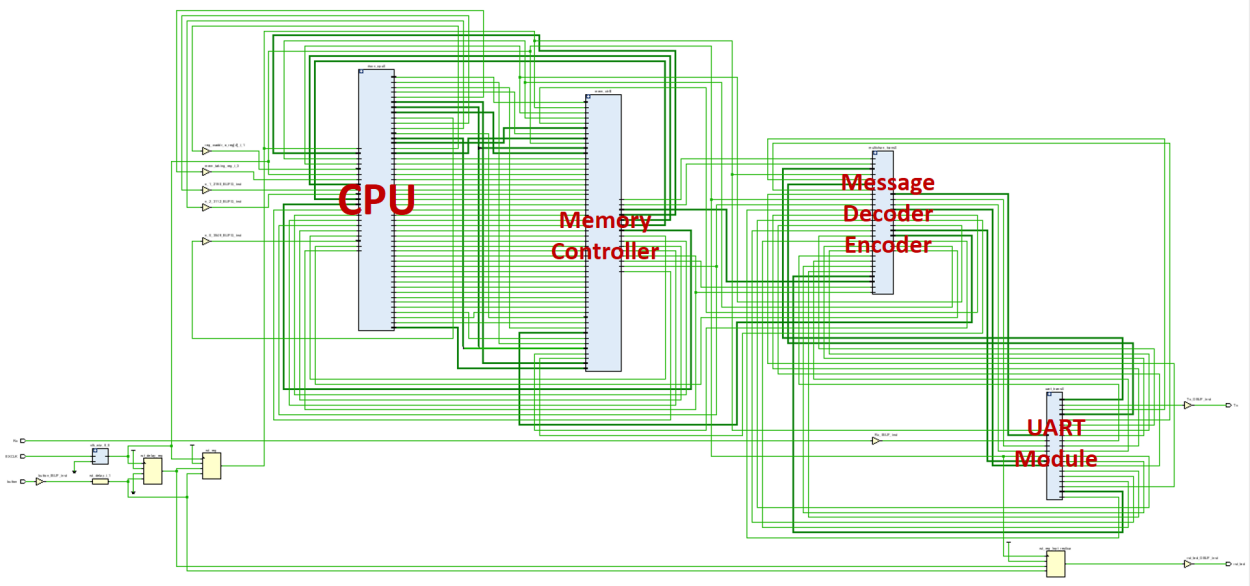


Fig. 3: Scematic Overview

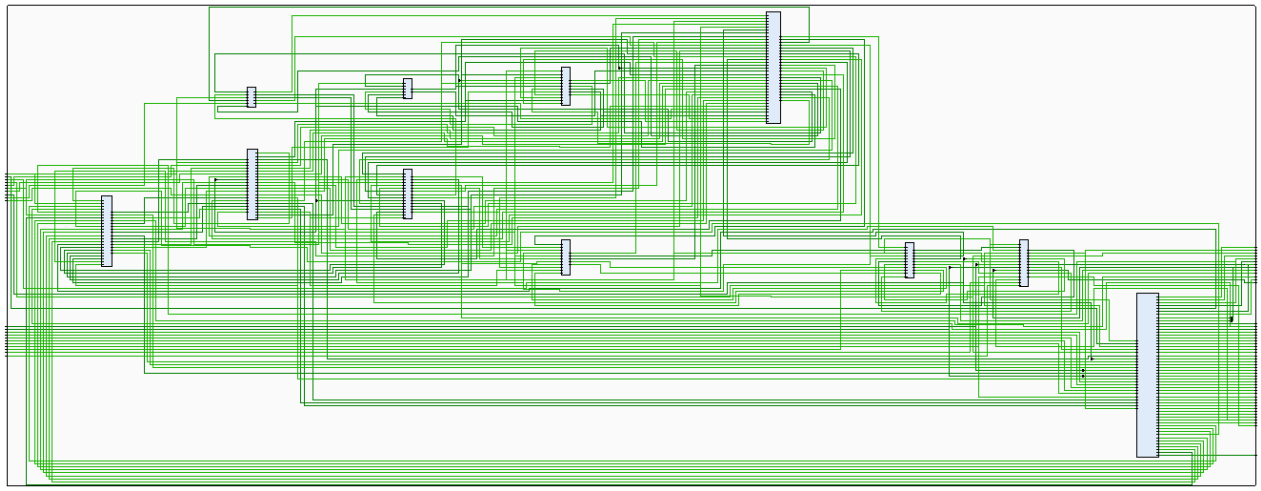


Fig. 4: CPU Module Scematic

References

[1] 雷思磊. *自己动手写 CPU*, 电子工业出版社, 2014.

[2] John L. Hennessy, David A. Patterson, et al. *Computer Architecture: A Quantitative Approach*, Fifth Edition, 2012.

[3] David A. Patterson. PPT of *CS252 Graduate Computer Architecture*, 2001.

[4] Zhekai Zhang's (张哲恺) MIPS CPU project. https://github.com/sxtyzhangzk/mips-cpu

[5] Chris Fletcher. *Verilog: always @ Blocks*