Project Report of RISC-V CPU

Course Project of MS108 Computer System(1)

Bohan Hou (侯博涵)
ACM honor class @ Shanghai Jiao Tong University

1 Introduction

GitHub repository of my CPU project: https://github.com/spectrometerHBH/Chaos

This project is a RISC-V CPU with Tomasulo Algorithm implemented in Verilog HDL, which is runnable on FPGA(tested on xc7a35t).

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
Dynamic Sceduling	Tomasulo Algorihm
Cache	512B direct indexed I-cache
Memory	128K bram memory on chip

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 programs. But something goes wrong when tested on FPGA (Basys 3), which is explained in the next section.

3 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 would goes through all stages before the next instruction is fetched. Though there is I-cache, it only works for loops. Therefore, UART may not be a perfect solution to memory simulation.

Experience with FPGA After cache and UART module passed simulation, I tested the CPU on Basys 3 FPGA, it read the first 5 instructions correctly but did not jump at the 5th instruction, which is a branch instruction. That test program did not go wrong in Vivado simulation. In the last two days before deadline I tried to find the bug but did not make it. Zhanghao Wu gave me a tutorial about always @ blocks[5], from which I learned about latch generation. My original design for stall control with IF stages caused some latch inferring, which could be "a terrible place for bugs"[5]. I re-designed part of code to avoid latches, and found that it may need some more debugging. From this experience, I learned that some hidden trouble would not reveal itself in simulation. And I should have learned more about Verilog HDL before starting to write code.

Automation To compile and assemble test programs RISC-V toolchain and some other tools are used. Many commands are needed in this process. I wrote a Makefile and found it really helpful. Automation improves efficiency and it feels good!

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.

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*