AIRA RISC-V CPU Manual

于峥 518030910437

2020 年 1 月 3 日

This project is a trivial RISC-V CPU with tomasulo algorithm implemented in Verilog HDL, which is a course project of Computer Architecture, ACM Class @ SJTU.

1 Design & Architecture

1.1 Environment

Device Name	Aira
ISA	RISCV 32I
FPGA	Basys3

1.2 Out-of-order Execution

The main feature of the Aira RISC-V is to support out-of-order execution, some algorithm used in the Aira RISC-V are brefly introduced in the table below:

Features	RISC-V CPU
Dynamic Scheduling	Tomasulo Algorithm
Piplining	3-stages pipeline (Fetch, Dispatch, Execute)
Multiple FU	2 Arithmetic/Logic Units and 1 Load/Store Unit

1.2.1 Details

- In order to achieve fetch one instruction four cycles, every cycle the fetcher will send pc and pc+4 to the I-cache.

- The fetcher receive instruction at negedge.
- For JAL instruction, fetcher calculate the target address and modify pc directly.
- Allocator rename register at dispatch stage.
- Every EX unit has a reserved station.

1.2.2 Design Diagram

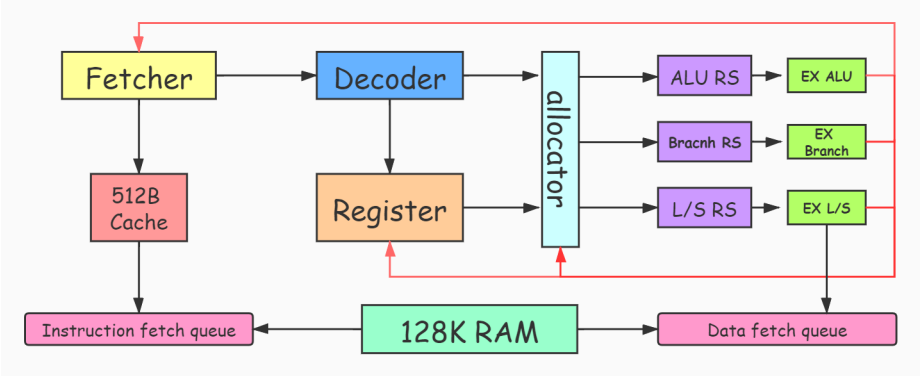

图 1: Aira RISC-V CPU design diagram

1.3 Cache

Parameter	Data
Type	Instruction Cache
Size	512B
Set Associative	2-way associative
Replacement Policy	FIFO

I write a direct-map D-cache in other branch, I deleted it because it didn't make the program faster.

1.4 Other Features

Using LED light for output and debug, and segment display as timer.

2 Performance

The highest frequency that the CPU can reach is 200MHz, but it is not always stable, so I choose 170MHz version to test, here are time consuming of some test points:

testcase	time(s)	IPC
pi	1.10	0.390
bulgarian	0.54	0.689
hanoi	0.76	0.783
queens	0.81	0.779

from the waveform, we can see multiple ex unit works at same time.

3 Remarks & Summary

CPU 大作业最大的缺点就在于需要写完整个CPU才能看到最后的成果，所以在一开始会写的很没有动力，并且由于对语言，硬件本身的不熟悉，导致很容易写出不那么优秀的代码。我在写完后, LUT(Look up table)的占用量达到了 170%, 并且wns达到了-2.8ns，所以我在保证正确性后几乎所有的时间都在研究如何减少wns，提升频率。

我wns为-2.8ns的主要原因是我使用了negedge，导致很多路径能够使用的时间只有5ns再100Mhz下，并且我在三个阶段都使用了negedge，如果想完全消除negedge只能重构代码，所以我只能先把dispatch阶段的negedge消掉，并且改小了cache, 使其wns减少到-0.9ns左右。后来我发现最后一个阶段做的事情似乎做的太少了，于是我把保留站移到了这个阶段工作，之后又精简了一些代码和信号的大小，并且把 vivado 的优化力度开到最大，终于达到了0.042ns。

之后我尝试加上了直接映射的D-Cache，不过优化效果并不明显，并且显著增加wns, 所以我没有把它并入我的最后版本。最后我尝试在CPU上运行一些更复杂的程序，例如操作系统。不过我发现这是否成功并不在于CPU，而在于RV32I指令集所能支持的事情很少，更多的工作可能需要花在编译操作系统上。而且我仅仅只是将浮点部分运算编译成汇编代码后所需要的内存空间就达到了RAM的1/4,所以我建议以后不要将这样一个不切实际的要求作为满分标准。

我最后玩了一下这个板的其他功能，例如上面的LED灯等，我认为大作业可以适当向这个方向发展，增加作业趣味性。

总的来说，我认为我的Tomasulo虽然并没有使用太多高级的特性，例如双发射等，但是我将每个部分的设计都尽量精巧，让tomasulo能够有较好得频率。

4 References

1. John L. Hennessy, David A. Patterson, et al. Computer Architecture: A Quantitative Approach, Fifth Edition, 2012.

2. 雷思磊. 自己动手写 CPU. 电子工业出版社, 2014.

3. RISC-V ISA Specification