RISC-V CPU 大作业说明文档

目录

RISC-V CPU 大作业说明文档
目录
1. 简介
2. 架构示意图及说明
示意图
说明
3. 一些细节上的实现
内存读写操作
取指令与访存的竞争
寄存器的同时读写
分支跳转指令的实现
指令缓存的实现
4. 遇到的问题
取指模块的暂停
inferring latch(锁存器)
Cache 被综合进bram
5. 大作业心得
6. 对于大作业的建议

1. 简介

本次大作业主要目的是实现一个基于riscv架构, rv32i指令集的cpu。本项目一共实现了以下内容：

- 取指-译码-执行-访存-回写 的五级流水架构
- FPGA 100MHz 测试通过
- 512Byte的指令缓存，可以容纳128条instruction，pi测试点时间为3.9s

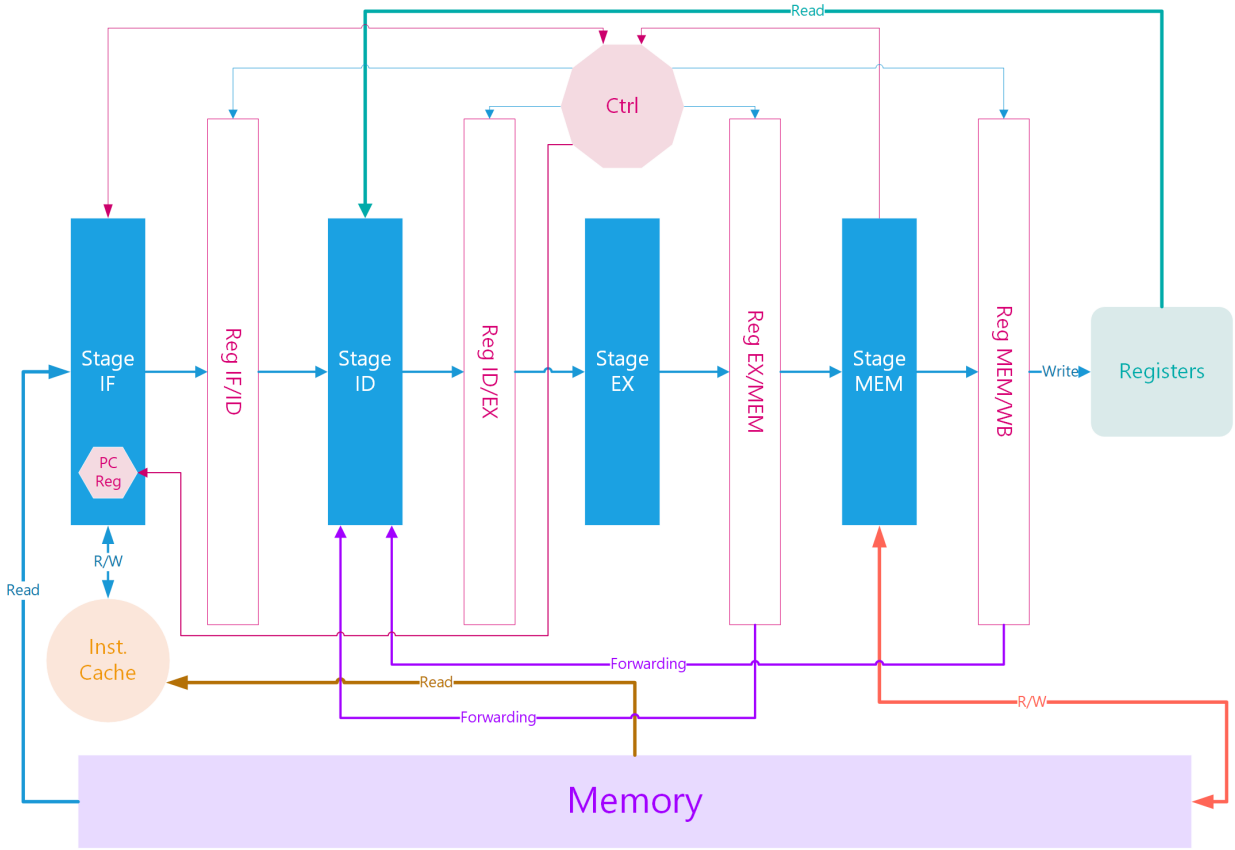
（注：FPGA上测试时需要将 `Synthesis Settings` 里面的 `Fanout Limits` 改为 `600` ）

本项目主要使用verilog硬件设计语言编写，使用Xilinx Vivado进行仿真测试和综合。

2. 架构示意图及说明

示意图

本项目设计的cpu架构的流程图大致如下图所示：



说明

图中各阶段的功能如下：

- stage_if：取指令模块，可以在6周期内从memory中取出一条指令，或者在2周期内从cache中取出一条指令。
- pc_reg：集成在stage_if取指令模块中，用作pc记录要取的指令在内存中的地址，会在每取完一条指令后自动加4，或者是根据跳转命令的目标地址进行跳转修改。
- Reg IF/ID：取指到译码阶段的流水线寄存器，用与五级流水的控制，传递指令到译码阶段。
- Stage_id：译码模块，可以将指令进行拆分，判断指令类型、提取指令中的立即数并且进行立即数扩展。这一阶段同时也将访问寄存器得到源操作数，或者根据forwarding数据线得到源操作数。
- Reg ID/EX：译码到执行阶段的流水线寄存器，用于五级流水的控制，传递译码器的指令类型、源操作数和目标寄存器地址(或目标内存地址)。
- Stage_ex：执行模块，接到译码模块的源操作数和指令类型之后开始运算，得到运算结果。
- Reg EX/MEM：执行到访存阶段的流水线寄存器，用于五级流水的控制，传递运算结果、目的寄存器地址或者目的内存地址，以及将结果forwarding到译码器。
- Stage_mem：访存模块，如果该指令不是LOAD或者STORE指令，该模块仅仅是将数据传递给下一个流水线寄存器。如果是LOAD或者STORE，访存模块会申请进行内存读写操作。本模块实现了6周期的Load Word和5周期的Store Word。
- Re MEM/WB: 访存到回写阶段的流水线寄存器，用于五级流水的控制，传递要回写的结果给寄存器模块，以及将结果

forwarding到译码器。

- Ctrl：控制模块，可以通过暂停流水线寄存器来实现流水的暂停。
- Inst Cache：指令缓存，用于暂存取到的执行，一边在第二次访问时能够更快地得到指令。。
- Registers：寄存器模块，实现了32个rv32i架构中的整数寄存器，提供两个读接口和一个写接口。。
- Memory：这部分是助教已经封装好的，总大小128KB，每次只能读或者写一个Byte的数据。。

3. 一些细节上的实现

内存读写操作

- 读：内存一次只能提供1个Byte的数据，而且为了保持稳定性，一般是将地址发给内存后，下一周期内存才会把数据提供出来，故需要控制实现多周期取数据。本项目在取指和访存模块中内嵌有限状态机，根据时钟信号在多个状态之间切换，通过 `发地址1-发地址2-取数据1/发地址3-取数据2/发地址4-取数据3-取数据4` 实现6周期取到一个Word。
- 写：写与读相差不太大，也是通过有限状态机控制。通过 `发地址1/发数据1-发地址2/发数据2-发地址3/发数据3-发地址4/发数据4-等待内存完成` 实现5周期写入一个Word。

取指令与访存的竞争

- 流水线架构不可避免地会出现Structure Hazard，取指令和访存阶段的竞争就是一个非常典型的Structure Hazard。本项目给出的解决方案是优先保证访存阶段的操作，当访存申

请发出后，ctrl模块会发出暂停信号，暂停流水线和取指令操作。

寄存器的同时读写

- 本项目中寄存器实现了边读边写，其实实现起来也非常简单，只需要在读取寄存器时判断一下是不是正在写这个寄存器就可以了。

```
.......
if ((raddr1 == waddr) && (we == 1'b1) &&
 (re1 == 1'b1)) begin
    rdata1 <= wdata;
end
.......
```

分支跳转指令的实现

- 本项目没有做分支预测，在取出一条指令后会判断是否为分支跳转指令，如果为分支跳转指令会给ctrl模块发送分支跳转暂停请求暂停取指令。同时在译码阶段会判断分支跳转是否发生并且得到正确的下一条指令地址，回传到取指模块改写pc寄存器，实现分支跳转指令。

指令缓存的实现

- 本项目实现了一个512Byte的instruction cache，一共是128行，每行包含 1-bit 的valid位、10-bits 的tag位、32-bits 的data位。在取指模块中开始取指时，会同时向内存和cache

同时发出地址，然后在下一周期cache会告诉取指模块是否命中以及指令数据。如果命中则终止内存访问，如果没有命中则等待内存取指完成后，将指令地址和指令数据发送给cache，在cache中作一份备份。

4. 遇到的问题

取指模块的暂停

- 取指模块在取内存数据时，取到的是上上周期发给内存的地址的数据。如果取指被访存阶段暂停，当访存完成后并不能直接读取内存给出的数据，要重新发送一边地址才行。

取指阶段的部分代码：

```
.......
4'b0011: begin
    if (stall_sign[1]) begin // 正在访存操作
        cnt <= 4'b1010;
    end
.......
end
.......
4'b1010: begin
    if (!stall_sign[1]) begin // 等待访存结束
        mem_addr_o  <= pc_o[16:0] + 17'h1; // 重新发送地址
```

```
            cnt             <= 4'b1011;
        end
    end
    4'b1011: begin
        mem_addr_o      <= pc_o[16:0] + 17'h
    2; // 重新发送地址
        cnt             <= 4'b0011; // 跳回到
    正常的取指
    end
```

inferring latch(锁存器)

- 如果在组合逻辑中，某些寄存器变量并没有在所有的逻辑情况中均出现，则会在综合时出现锁存器:
- 例如:

```
reg a;
reg b;
always @(*) begin
        if (reset) begin
            a       <= 1'b0;
            b       <= 1'b0;
        end else begin
            a       <= 1'b1;
        end
    end
```

综合时会出现

`[Synth 8-327] inferring latch for variable 'b_reg'`

- FPGA上并没有锁存器，但是综合时会使用非常复杂的方式实现锁存器结构。这样会增加延迟，并且会造成布线Timing计算不准确。
- 应该尽量避免使用锁存器，但是有些时候必须要用的话也不是不可以。

Cache 被综合进bram

- FPGA上有一段bram空间，专门用来储存数据，但是这一部分空间访问延迟比较大，不符合cache的要求，可以在cache模块的定义中加入 `(* ram_style = "registers" *)` 使cache被综合成寄存器

```
(*ram_style = "registers"*) reg[31:0] ca
che_data[127:0];
(*ram_style = "registers"*) reg[9:0]  ca
che_tag[127:0];
(*ram_style = "registers"*) reg       cac
he_valid[127:0];
```

5. 大作业心得

这次cpu大作业过后，我自己感觉主要有以下收获：

- 更加熟悉了五级流水的结构，通过硬件设计使得自己对流水线架构的认识更清晰

- 学习到了一种全新的硬件编程思路，自己的编程能力也得到了一定的提升
- 对于数字电路有了一定的了解
- 将体系结构课程中的一些知识点运用到实践当中，对于体系结构知识点的认识更加深刻

6. 对于大作业的建议

- 建议助教开放内存的限制，允许我们自己修改内存，这样的话应该可以实现更高性能的内存，从而实现更高性能的cpu。