**体系结构课程第 二 次实报告**

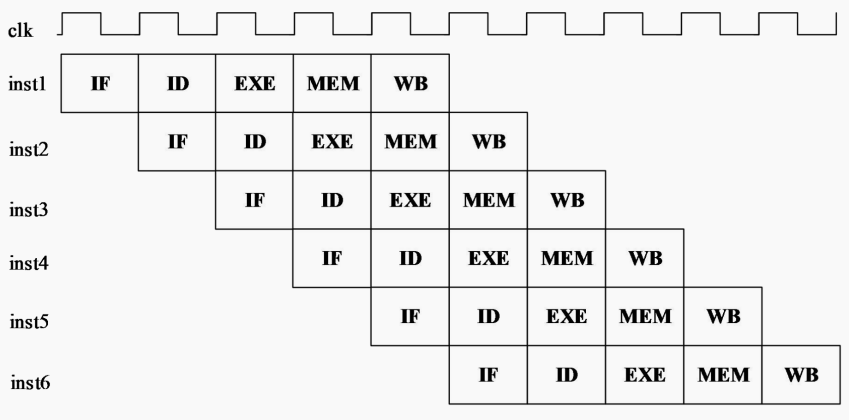
**实验名称：静态 5 级流水线 CPU 实现**

学号： 2113203 姓名： 付政烨 班次： 0855

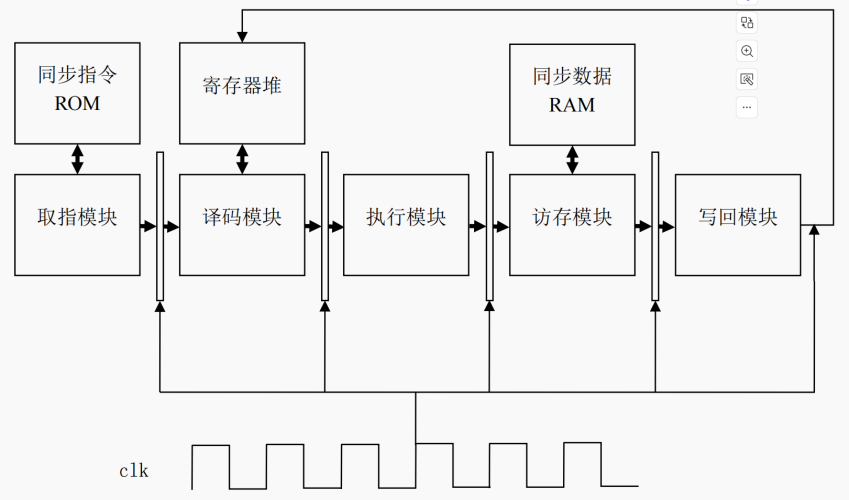
1. **实验目的**
2. 在多周期 CPU 实验完成的提前下，深入理解 CPU 流水线的概念。
3. 熟悉并掌握流水线 CPU 的原理和设计。
4. 最终检验运用 verilog 语言进行电路设计的能力。
5. 通过亲自设计实现静态 5 级流水线 CPU，加深对计算机组成原理和体系结构理论知识的理解。
6. 培养对 CPU 设计的兴趣，加深对 CPU 现有架构的理解和深思。
7. **实验内容说明**

请结合实验指导手册中的《静态五级流水线 CPU》实验，完成功能验证并进行如下工作：

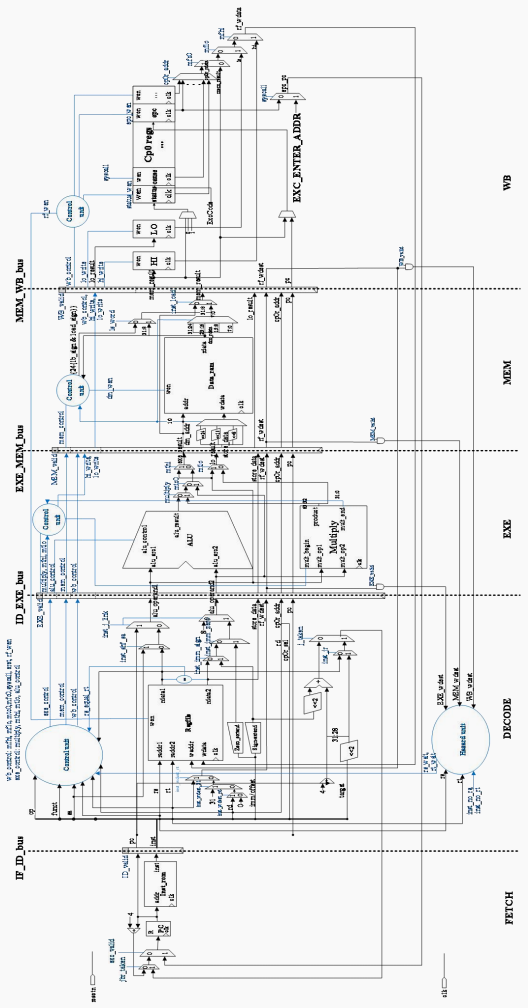
1. 分析现有指令的执行过程，找到存在的读后写/写后读问题，并进行改进验证（推荐自己编写指令 rom 中的指令，针对存在的问题进行测试改进）然后写入实验报告。
2. 在做 1 的过程中，寻找现有 CPU 的不足之处，提出一些自己的改进方案和想法，包括但不限于实验指导手册第十章中的优化部分，并初步讨论一下可行性，为最后一次实验做准备。
3. 实验报告中的原理图放图 9.2 即可。注意要在实验报告中详细说明改进过程，验证时先描述改进前的情况，再说明改进后的情况，进行具体对比。
4. **实验原理图**



理想流水线 CPU 时空图

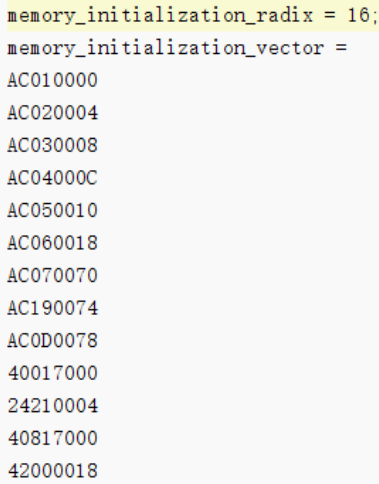


静态 5 级流水 CPU 大致框图



5 级流水线 CPU 的实现框图

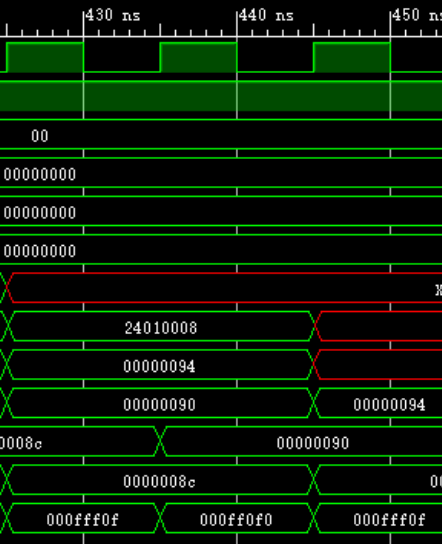
1. 实验步骤
2. **生成 IP 核**：在设计基于 FPGA 或 SoC 的系统时，IP 核的生成是实现硬件加速和系统功能模块化的重要步骤。根据实验教程的指导，首先需要利用 Xilinx 提供的库来实例化 RAM 和 ROM 的 IP 核。具体而言，可以通过 Xilinx Vivado 工具进行 IP 核的生成。在生成 ROM IP 核时，必须加载相应的初始化文件（通常为 .coe 文件）。该文件包含 ROM 中存储的数据或指令集，是 ROM 初始化过程中的关键元素。

**生成 IP 核 导入源码**

1. **导入源码**：完成 IP 核的生成后，接下来需将设计源码导入到项目中。导入过程包括将编写的 Verilog 代码，以及生成的 IP 核文件正确集成到项目目录中。此步骤的关键在于确保所有源码文件与 IP 核文件的路径正确，且编译设置与设计要求相匹配，从而确保在后续的仿真和实现阶段，所有模块能够顺利连接与协同工作。
2. **实验结果分析**
3. **问题分析（改正前）**

在程序运行过程中，当程序计数器（PC）值为 00000094 时，程序出现异常。通过查阅实验指导手册表 9.5，发现该地址对应的指令为 addiu $1, $0, #8（将立即数 8 加到寄存器 $0 的值中，并存储到寄存器 $1）。进一步分析发现，在 PC=00000090 时执行的指令为 jalr $27（跳转并链接寄存器），程序应在此跳转到地址 50H，但实际运行中并未正确跳转，证明问题在此位置出现。通过对波形的观察发现，异常的根本原因是写后读（Write-After-Read, WAR问题。在 PC=0000009C 时，程序对寄存器 $27 执行写入操作，但紧接着在 PC=00000090 时需要读取寄存器 $27 的值，用于跳转地址的计算。由于寄存器的写入操作尚未完成，导致读取的值不正确，从而引发程序异常。



1. **问题修正（代码改正）**

为解决上述问题，设计了一种方法，通过在 IF（取指周期） 中增加一个寄存器延迟来引入额外的时钟周期。在每次应当执行取值周期时，先等待一拍，再执行指令取值。这样可以确保在遇到写后读的情况时，寄存器的写入操作已经完成，从而保证后续读取的值是正确的。具体修改包括：

* 在 IF 阶段增加一个额外的锁存器，用于存储当前指令，延迟一个时钟周期后再发送至后续流水段。
* 在流水线控制逻辑中，添加相应的等待机制，使数据冒险（Data Hazard）得到充分处理。

reg wait\_cycle; // 新增寄存器，用于等待一拍

always @(posedge clk)

begin

if (!resetn || next\_fetch)

begin

IF\_over <= 1'b0; // 重置等待周期

wait\_cycle <= 1'b0;

end

else if (wait\_cycle)

begin

IF\_over <= IF\_valid; // 在等待周期后执行取指有效信号

wait\_cycle <= 1'b0; // 清除等待标志

end

else

begin

wait\_cycle <= 1'b1; // 设置等待周期

IF\_over <= 1'b0; // 仍然保持 IF\_over 为 0

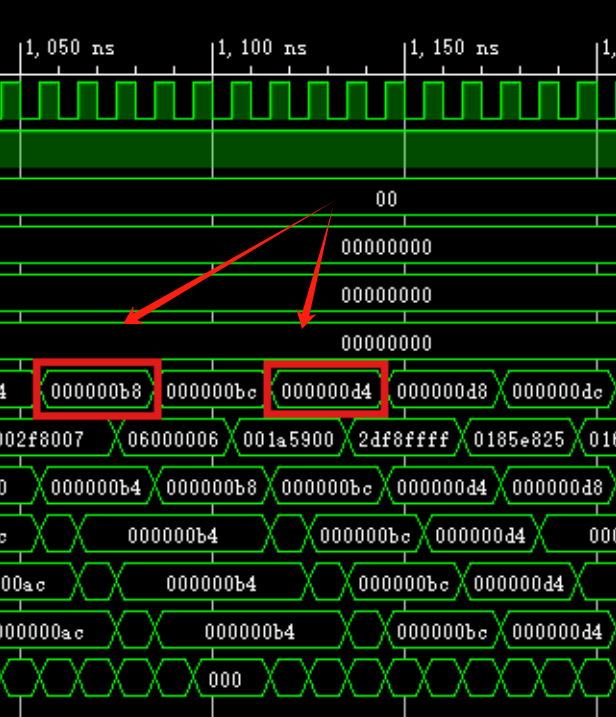
end

end

1. **修正结果（改正后）**

在程序修正后，对潜在可能发生同样写后读问题的部分进行全面检查和处理。通过波形仿真结果验证了修改效果，程序在运行至 PC=B8 时，成功跳转至 PC=D4，且后续功能运行正常，未再出现异常。由此证明，此次修改有效解决了写后读问题。然而，通过进一步分析波形发现，CPU 的流水线效率存在一定问题。例如，信号 cpu\_5\_valid 在运行过程中多数时间保持为 0xfffff，即流水线经常处于等待状态。虽然这种方式能够确保正确性，但效率较低。

为进一步提升流水线效率，可以尝试引入旁路机制（Bypass/Forwarding Mechanism），即在寄存器尚未写入完成时，通过直接转发执行阶段或内存阶段的数据结果，避免等待额外的一个时钟周期。这种方法可以显著减少流水线暂停时间，提高整体运行效率，同时保证程序的正确性。



综上，修正代码的策略有效解决了写后读问题，并提供了明确的改进方向：在确保正确性的基础上，通过优化流水线数据传递方式（如旁路机制）进一步提升处理器性能。

1. **总结感想**

通过本次实验，我深入理解了静态五级流水线 CPU 的设计原理和实现方法，并掌握了 Verilog 语言在硬件设计中的应用。实验过程中，分析并解决了写后读数据冒险问题，验证了通过引入寄存器延迟的方式能够有效提高程序的正确性。同时，我意识到当前解决方案虽然确保了正确性，但流水线效率较低。为此，我提出了引入旁路机制以优化数据传递的改进方案，进一步提升性能。本次实验不仅巩固了对计算机组成原理和体系结构的理解，还锻炼了实际动手能力，加深了我对 CPU 设计的理解。