**体系结构课程第 三 次实报告**

**实验名称：期末综合实验**

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。
6. **实验内容说明**
7. 分析多周期CPU是如何改进成五级流水线CPU的。
8. 在五级流水线CPU的代码中也存在类似的时钟问题/分支跳转问题，请将此问题进行分析修复，并总结修复过程。
9. 在现有基础上，尝试对CPU进行改进，如旁路，分支预测等内容。
10. 进行波形仿真。
11. 进行上箱验证
12. **实验原理**

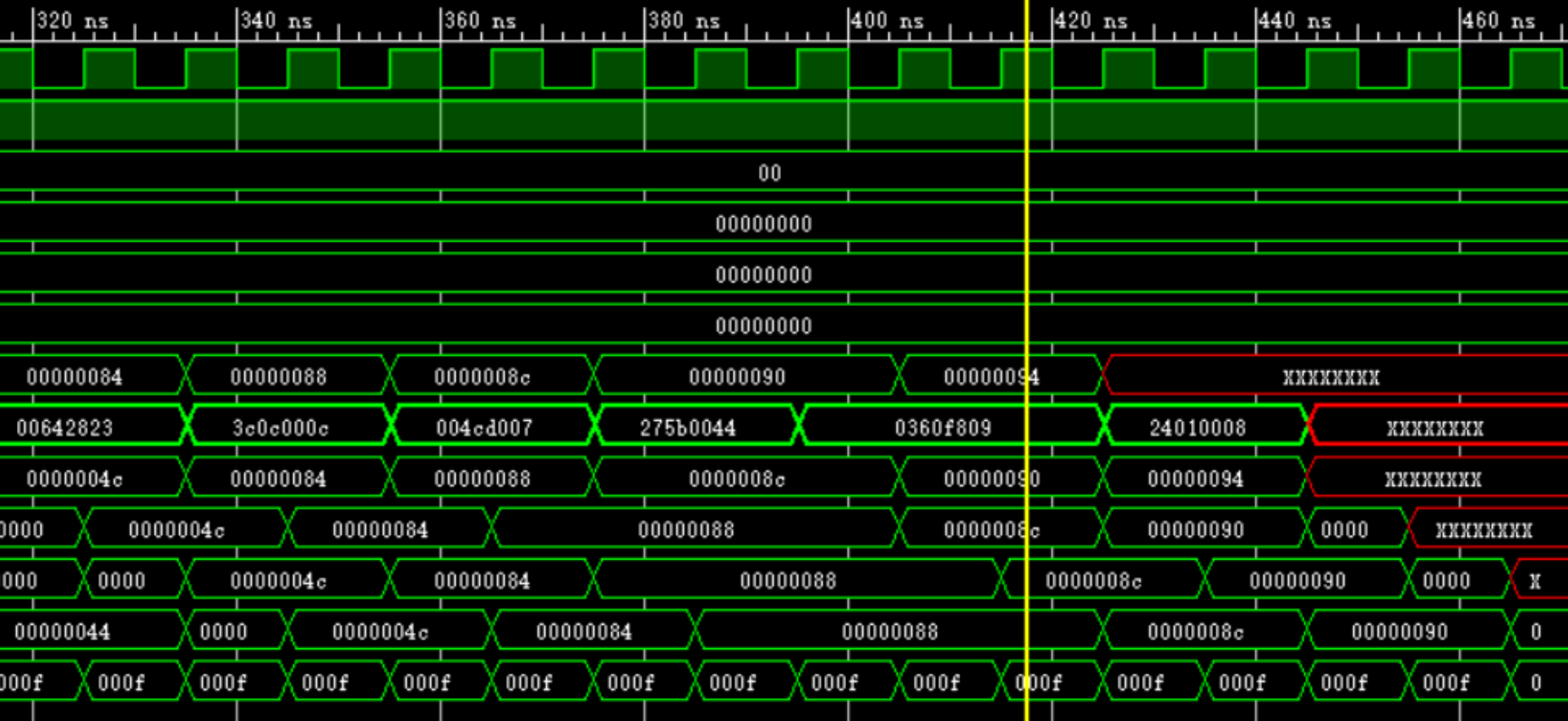
在计算机架构中，多周期CPU通过将指令执行过程分解为多个时钟周期来完成每条指令的执行。在每个时钟周期内，CPU执行不同的操作，如取指、译码、执行、访存和写回等。相比之下，五级流水线CPU则将指令执行过程划分为五个独立的阶段，并将每个阶段安排在不同的时钟周期内并行执行。这种设计使得多个指令能够在同一时刻处于不同的执行阶段，从而显著提高了CPU的吞吐量和整体效率。

将多周期CPU改进为五级流水线CPU的过程涉及多个方面的设计改进。首先，五级流水线CPU需要增加更多的控制信号来管理各个流水线阶段的操作。这些信号除了传统的有效信号（valid）外，还包括“完成信号”（over）、允许输入信号（allow\_in）和取消信号（cancel）。例如，译码阶段在允许进入执行阶段之前，必须确保译码操作已完成，并且该阶段的硬件资源可以为即将进入的指令提供支持。此外，流水线阶段的控制信号会根据前一级的状态进行调整，从而确保流水线的顺利运行。其次，为了应对异常处理和中断请求，五级流水线CPU增加了多个异常信号和接口，如exc\_bus、exc\_valid等。这些信号能够在发生异常时及时通知CPU并更新PC寄存器至异常入口地址，保证指令执行的正确性和系统的稳定性。此外，为了增强流水线的灵活性，五级流水线设计引入了新的变量，例如EXE\_wdest、MEM\_wdest和WB\_wdest，这些变量用于存储不同流水线阶段的目标寄存器位置。这些寄存器能够在指令执行过程中动态调整，确保数据能够在各个阶段之间正确流动，进一步提升执行效率。

在多周期CPU中，指令执行的判断条件较为简单，通常通过在每个时钟周期的上升沿更新IF\_over信号来控制指令流转。然而，五级流水线设计需要更复杂的控制逻辑，因为每个阶段的执行存在一定的顺序依赖性。在五级流水线中，IF\_over信号需要在指令取指阶段完成后才会置为有效，同时通过锁存机制将其与其他阶段的状态进行协调。此外，流水线设计还需要保证指令的顺序性，避免指令间的冲突与依赖问题，特别是在源寄存器和目的寄存器的写回操作中。为了有效解决流水线中的数据相关性问题，五级流水线CPU设计增加了对数据冒险的处理机制。若指令中存在源或目的寄存器，但这些寄存器正在被后续流水线阶段写回，则当前指令需等待前述寄存器的更新完成。在这种情况下，rs\_wait信号会被置为1，表示当前指令暂时无法执行。对于跳转和分支指令，只有在指令的取指阶段完成后，才可进入译码阶段并进一步执行，否则会导致程序计数器（PC）更新错误，影响程序的控制流。最后，五级流水线CPU需要处理与分支指令相关的延迟槽问题。在某些情况下，跳转指令会造成控制流的不确定性，导致后续指令在跳转前的执行不确定。为了解决这个问题，五级流水线设计要求在链接跳转时计算跳转指令后的PC地址，并将其存入特定的寄存器（如31号寄存器）中，以确保跳转操作的正确与高效。

1. **实验步骤**
2. **问题分析（改正前）**

在实验中，首先将汇编代码的机器指令存入只读存储器（ROM）中，并进行仿真运行。通过仿真，发现当指令执行到jalr $27时，程序无法正确地获取下一条指令。例如，指令0360F809对应的jalr $27指令与延迟槽指令24010008（即addiu $1, $1, 8）存在一定的执行顺序问题，导致程序在该指令处停滞，无法继续运行。



在五级流水线CPU的执行过程中，可能会出现数据冒险，尤其是“写后读”问题，导致指令结果不符合预期。实验中的错误可归结为以下两个原因：

1. **指令执行错误**：对于指令$3CH addu $3, $2, $1，执行结束后，寄存器$3应保存结果0011H，但仿真结果显示不正确。这是因为$1和$2的初值为0，导致加法运算的结果为0，而不是预期的10 + 1 = 11。
2. **跳转指令的错误：**对于指令jalr $27，已知地址为8C的指令为addiu $27, $26, 68，接着是指令jalr $27。这两条指令存在数据依赖关系：在执行jalr之前，必须首先将$27寄存器中的跳转地址更新。然而，在实验仿真中，指令jalr未等待addiu指令的写回阶段完成，导致跳转目标地址错误，进而导致程序执行失败。
3. **问题修正（代码改正）**

为了修复上述错误，优化实验中的五级流水线设计，采取了以下几项改进措施：

**a. 修改fetch.v和mem.v**

在流水线CPU中，存在类似于多周期CPU的问题，特别是在处理load指令时。为此，修改了指令取取阶段（fetch）和存储阶段（mem）的时序，使得每个阶段的执行延时延长为3个时钟周期。这样能够有效解决因数据冒险导致的读取延迟问题，特别是在load指令执行时，需要保证数据已经从内存加载完毕。

|  |  |
| --- | --- |
| **修改fetch.v** | **修改mem.v** |
| reg stall;  always @(posedge clk)  begin  IF\_over <= (!resetn || next\_fetch) ? 1'b0 : (stall ? IF\_over : IF\_valid);  stall <= (!resetn || next\_fetch) ? 1'b0 : (stall ? 1'b1 : stall);  end | reg MEM\_valid\_r;  reg stall;  always @(posedge clk)  begin  MEM\_valid\_r <= MEM\_allow\_in ? 1'b0 : (stall ? MEM\_valid\_r : MEM\_valid);  stall <= MEM\_allow\_in ? 1'b0 : (stall ? 1'b1 : stall);  end |

**b. 引入旁路技术**

为了解决数据冒险和“写后读”问题，本实验通过旁路技术实现了数据的前递。在流水线的译码（ID）阶段，增加了对EXE、MEM和WB阶段执行结果的输入。具体来说，在decode模块中，通过判断寄存器rs和rt是否为当前正在写回的寄存器（即EXE\_wdest、MEM\_wdest、WB\_wdest），来决定是否将上一个阶段的计算结果前递到当前阶段。

|  |
| --- |
| **数据冒险检测与阻塞信号生成** |
| // -----{ID执行完成}begin  // 检查数据冒险  wire rs\_wait;  wire rt\_wait;  assign rs\_wait = (~inst\_no\_rs && (rs != 5'd0)) &&  ((rs == EXE\_wdest) || (rs == MEM\_wdest) || (rs == WB\_wdest));  assign rt\_wait = (~inst\_no\_rt && (rt != 5'd0)) &&  ((rt == EXE\_wdest) || (rt == MEM\_wdest) || (rt == WB\_wdest)); |

通过检测rs\_wait和rt\_wait信号，判断是否存在数据依赖关系，若发生数据冒险，则进行数据前递操作。这保证了后续指令能够正确读取到已经更新的数据，避免了因“写后读”导致的错误。

**c. 修改decode.v模块**

在decode.v模块中，新增了EXE\_result、MEM\_result和WB\_result作为输入信号，以便将上一阶段的结果传递到译码阶段。这样，译码阶段的指令可以根据这些结果判断是否需要进行旁路前递，保证数据一致性。

|  |
| --- |
| **数据冒险检测与译码模块** |
| module decode(  // 新增输入端口，用于数据冒险检测  input wire [31:0] EXE\_result,  input wire [31:0] MEM\_result,  input wire [31:0] WB\_result,  // 输出端口，用于展示当前PC值  output wire [31:0] ID\_pc  ); |

**d. 修改pipeline\_cpu.v模块**

在pipeline\_cpu.v中，为了支持新的数据前递机制，需要对数据通路进行修改。特别是增加了EXE\_MEM\_bus和MEM\_WB\_bus来传递各流水线阶段的执行结果。对于写回阶段的结果（WB\_result），需要从rf\_wdata中提取出来，进行后续处理。

|  |
| --- |
| **译码级实例化** |
| // 修改传入的参数位数等  decode ID\_stage (  // 译码阶段模块  .EXE\_result (EXE\_MEM\_bus[31:0]),  .MEM\_result (MEM\_WB\_bus[31:0]),  .WB\_result (rf\_wdata),  .ID\_pc (ID\_pc)  ); |

**e. 完善前递机制与数据一致性**

在decode.v模块中，通过引入前递技术（forwarding），确保在数据冒险发生时，可以将前一个阶段的计算结果传递到当前阶段。通过rs\_wait和rt\_wait的状态判断，系统能够在写回阶段完成数据更新后，进行相应的数据前递，从而避免出现数据一致性问题。

|  |
| --- |
| **数据前递赋值逻辑** |
| assign rs\_value = (rs\_wait && rs != 0) ?  (rs == WB\_wdest ? WB\_result :  (rs == MEM\_wdest ? MEM\_result :  (rs == EXE\_wdest ? EXE\_result : rs\_value)))  : rs\_value;  assign rt\_value = (rt\_wait && rt != 0) ?  (rt == WB\_wdest ? WB\_result :  (rt == MEM\_wdest ? MEM\_result :  (rt == EXE\_wdest ? EXE\_result : rt\_value)))  : rt\_value; |

**f. 调整流水线阶段的时序**

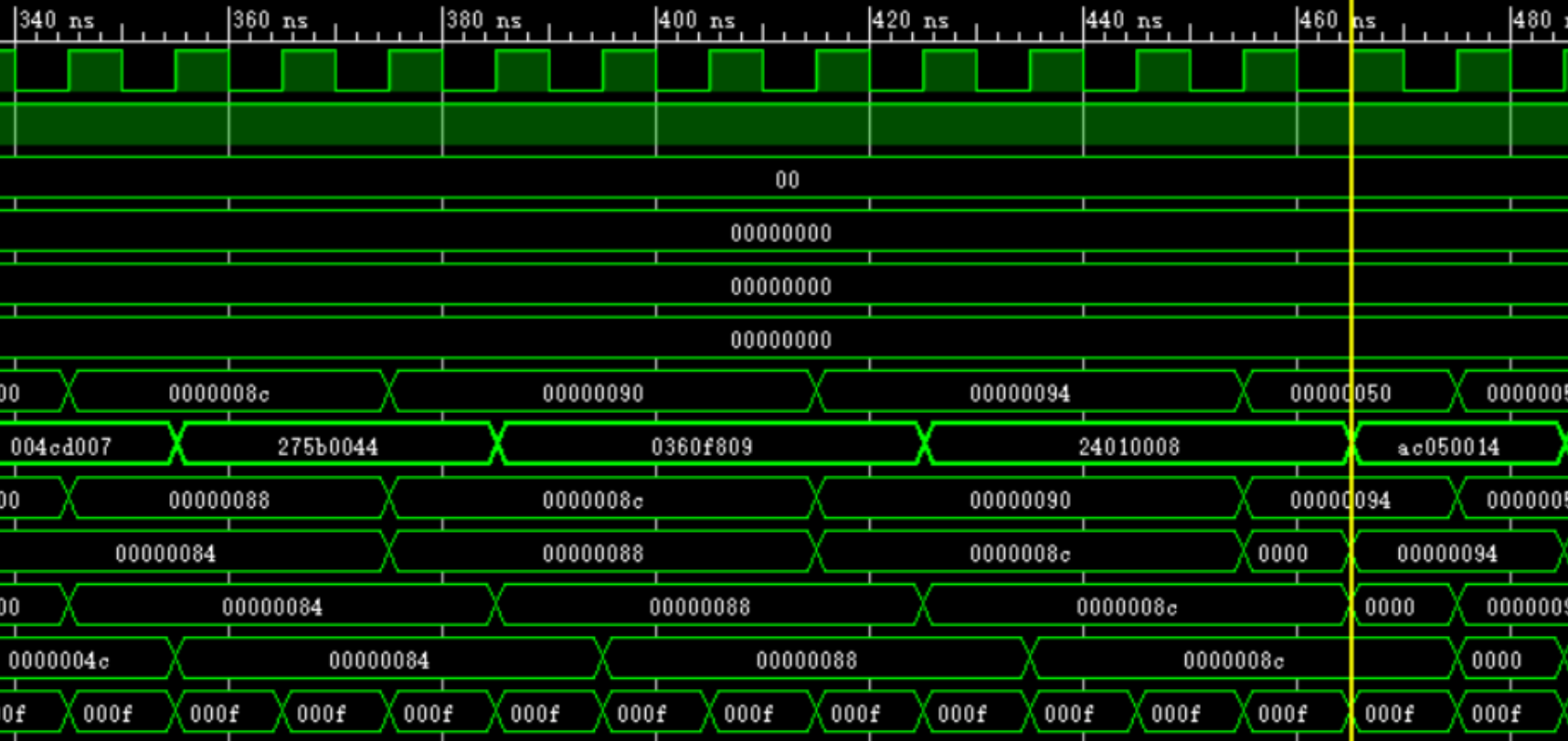
为了确保流水线能够正确处理数据冒险，特别是在跳转指令和load指令的情况下，调整了流水线各阶段的时序，确保每个阶段的执行能够正确等待前一阶段的结果，从而避免控制流错误和数据冲突。

通过以上一系列的实验改进，有效解决了五级流水线CPU中的数据一致性问题，特别是“写后读”问题，进一步提高了指令执行的准确性和流水线的效率。通过引入旁路技术和调整流水线的时序，能够在复杂的指令执行过程中确保数据的正确性，减少了流水线停顿和错误发生的概率，从而使得CPU的性能得到了优化。

1. **实验结果分析**

基于上述汇编语言程序的设计，进行了仿真验证，并成功生成相关波形。验证过程中发现了一些问题，经分析和修正后，程序在跳转指令的位置（具体为指令地址 0360F809 之后）可以正常继续执行。

修正后的程序成功实现了跳转操作，跳转偏移量为 50H（十六进制），对应的跳转指令为 AC050014。该指令通过有效的跳转偏移量和目标地址计算，确保程序流正确地转移至目标位置，从而验证了跳转指令的功能正确性及程序的连贯性。



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **跳转偏移量** | **指令** | **目标内存地址** | **机器码** | **二进制编码** |
| 50H | sw $5, #20($0) | Mem[0000\_0014H]=0000\_000DH | AC050014 | 1010\_1100\_0000\_0101\_0000\_0000\_0001\_0100 |

1. **改进实验其他部分潜在思路**
2. **提高流水线级数的优化探讨**

在现有五级流水线的基础上，可以考虑细化流水线阶段，将复杂操作分解为更小的子阶段。例如，将访存阶段进一步细分为地址生成阶段和实际访存阶段，将执行阶段细分为简单运算和复杂运算阶段。这种方式可以提高指令的并行度和整体吞吐量。然而，流水线级数的增加会带来额外的设计挑战，包括：

* **信号延迟的累积**：更长的流水线需要更多的锁存器来连接各级，这会引入额外的时序延迟，可能限制最高时钟频率。
* **预测失败惩罚增大**：分支预测错误时需要清空更多的流水线级数，导致更大的性能损失。
* **旁路设计复杂化**：级数增加后，数据前递路径变得更复杂，需要在多个阶段间传递结果，从而提高硬件实现的复杂性。

在增加流水线级数时，应综合考虑性能收益与设计复杂度之间的权衡。可以通过仿真分析不同流水线级数对CPI的影响，选择最优的设计方案。

1. **分支预测优化**

分支预测是影响流水线性能的关键技术，改进分支预测精度能够有效减少流水线停顿。常见的优化方法包括：

* **静态分支预测**：基于固定的规则，例如总是预测分支不发生，简单易实现，但精度较低。
* **动态分支预测**：通过硬件记录程序运行时的分支行为历史，预测未来分支的方向。
* **饱和计数器**：使用两位或四位计数器记录分支历史，提高预测的稳定性。
* **局部历史表**：为每个分支维护独立的历史记录，适合局部性强的分支行为。
* **全局历史记录**：维护整个程序的全局分支历史，适合分支行为与全局状态相关的程序。

改进分支预测的同时，还需优化跳转目标缓冲（BTB）和返回地址栈（RAS）的设计，确保在预测正确时能够快速提供目标地址，进一步减少延迟。

1. **数据相关冲突优化**

数据相关冲突是导致流水线停顿和性能下降的重要因素，可通过多种技术进行优化。旁路技术通过快速路径在流水线中直接获取后续指令的操作数，避免等待写回寄存器后再读取，实验中的基本旁路机制可以进一步优化，例如自动生成旁路信号以减少人为配置错误，并针对load-use冒险等特殊情况优化旁路逻辑。寄存器重命名通过引入硬件寄存器映射层，实现动态寄存器分配，能有效避免写后读（RAW）和写后写（WAW）冲突。乱序执行允许指令在硬件中动态重排序，只要保持程序语义不变即可，但需要增加指令调度窗口和寄存器重命名机制。此外，通过优化延迟槽指令，在跳转指令后填充对性能影响较小的指令，可以减少延迟槽空置引起的流水线停顿，从而进一步提升性能。

1. **异常处理与中断优化**

流水线CPU在处理中断和异常时需要清空部分流水线级数，这会导致性能下降。为提升效率，可以采用以下优化措施：首先，引入精准异常模式，在发生异常时记录异常指令的PC和流水线状态，确保异常处理的准确性并避免错误传播；其次，采用延迟异常机制，允许异常指令完成流水线中较短的部分操作后再处理，从而减少性能损失；最后，通过中断优先级管理，利用硬件优先级编码快速识别和处理最高优先级的中断，缩短中断响应时间并提高系统的实时性和稳定性。

1. **多核协作与共享资源优化**

在多核环境中，五级流水线CPU的性能优化可以通过改进多核间的共享资源管理来实现。首先，采用高效的缓存一致性协议是关键，例如MESI协议，可以有效解决多核间缓存数据一致性问题，避免因数据不一致导致的性能下降。其次，优化共享总线的仲裁机制，通过引入动态优先级分配或分布式总线架构，减少多核间的资源争用，提高总线带宽利用率。最后，加强核间通信的效率，通过设计低延迟的通信通道（如专用互连网络或点对点通道）以及优化消息传递协议，可以显著降低多核同步和数据共享的开销，从而提升系统的整体性能和并行效率。这些优化措施能够进一步释放多核体系结构的潜力，为复杂计算任务提供更强大的支持。

1. **总结感想**

通过本次的两次实验以及期末综合实验，我深入理解了多周期CPU与五级流水线CPU的架构原理及其优化设计过程。通过分析流水线中的数据相关冲突和分支预测问题，并借助旁路技术、数据前递机制及优化的控制逻辑，不仅成功解决了实验中的关键问题，还深刻体会到计算机体系结构设计的复杂性与艺术性。实验过程中，代码的不断调试与硬件行为的仿真让我对Verilog语言和CPU设计方法有了更加全面的掌握，同时也强化了理论与实践结合的重要性。

此外，通过这次实验，我进一步理解了流水线优化对系统性能提升的核心意义，也体会到在多核环境下，设计高效的资源共享机制和通信策略的重要性。这些宝贵的经验和收获离不开任课老师李老师和董老师的悉心指导。无论是课堂上的深入讲解，还是实验过程中的耐心解答，都为我提供了强有力的理论支持和实践方向。在此，我衷心感谢两位老师对我们的辛勤付出和无私奉献，也期待在未来的学习和研究中，能将这份知识和热情延续下去，为计算机体系结构领域做出更多贡献！