**北京邮电大学课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | **计算机组成原理课程设计** | | **学 院** | **计算机学院** | **指导教师** | **杨秦** |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **308** |  | **2023212122** | | **李思琪** |  | |
| **303** |  | **2023210710** | | **陈子容** |  | |
| **303** |  | **2023212872** | | **计子毅** |  | |
| **305** |  | **2023212304** | | **鹿一凡** |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | 本次课程设计的主要内容：设计五级流水线控制器，具有高效安全的处理数据冒险、控制冒险机制，创新RAC硬件栈实现以及预测返回地址检查和恢复错误机制，  分工：  李思琪：负责RAS（返回地址栈）预测机制的设计与实现，包括预测算法、错误检测与恢复机制的设计，以及数据冒险逻辑的修正和优化。同时负责技术方案报告的撰写和整体架构的设计。  陈子容：负责Y86流水线处理器的核心Verilog代码编写，包括五级流水线各阶段的实现、冒险处理逻辑的编码，以及系统的调试和安全性优化。同时负责实现文档的撰写和代码规范的制定。  计子毅：负责硬件平台的调试和优化，包括Vivado开发环境的配置、实验板管脚规范的研究、模式切换功能的实现，以及系统时钟的优化设计。同时负责硬件部分文档的撰写。  鹿一凡：负责硬件调试和用户界面的实现，包括LED显示系统的设计、周期计数数码管显示功能的实现，以及测试用例的硬件验证。同时负责硬件部分文档的撰写。 | | | | | |
| **学生**  **课程设计**  **报告**  （附页） |  | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | 遵照实践教学大纲并根据以下四方面综合评定成绩：  1、课程设计目的任务明确，选题符合教学要求，份量及难易程度  2、团队分工是否恰当与合理  3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果  4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范  **评语**:        **成绩**:  指导教师签名：  2025年 7 月 6 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

# 计算机组成原理课程设计实验报告

## 基于RAS预测优化的Y86-64五级流水线处理器设计与FPGA实现

学生姓名： 李思琪、陈子容、计子毅、鹿一凡

学号： 2023212122、2023210710、2023212872、2023212304

学院： 北京邮电大学计算机学院

指导教师： 杨秦、裴颂伟

完成日期： 2025年7月4日

**目录**

[计算机组成原理课程设计实验报告 3](#_Toc202824540)

[基于RAS预测优化的Y86-64五级流水线处理器设计与FPGA实现 3](#_Toc202824541)

[1. 需求分析 7](#_Toc202824542)

[1.1 项目背景与目标 7](#_Toc202824543)

[1.2 功能需求分析 7](#_Toc202824544)

[1.2.1 基础功能需求 7](#_Toc202824545)

[1.2.2 高级功能需求 7](#_Toc202824546)

[1.2.3 平台适配需求 7](#_Toc202824547)

[1.3 非功能需求分析 8](#_Toc202824548)

[1.3.1 性能需求 8](#_Toc202824549)

[1.3.2 资源需求 8](#_Toc202824550)

[1.3.3 可靠性需求 8](#_Toc202824551)

[1.4 成员分工 8](#_Toc202824552)

[2. 硬件环境 9](#_Toc202824553)

[2.1 实验板概述 9](#_Toc202824554)

[2.2 本实验用到的板卡资源 10](#_Toc202824555)

[2.2.1 LEDs（红、黄、绿各8个） 10](#_Toc202824556)

[2.2.2 按键开关 10](#_Toc202824557)

[2.2.3 7段数码管 10](#_Toc202824558)

[3. 实验原理 13](#_Toc202824559)

[3.1 Y86指令集体系结构 13](#_Toc202824560)

[3.1.1 程序员可见状态 13](#_Toc202824561)

[3.1.2 指令集设计原理 13](#_Toc202824562)

[3.1.3 指令分类与编码 14](#_Toc202824563)

[3.2 五级流水线架构设计 14](#_Toc202824564)

[3.2.1 流水线设计理论基础 14](#_Toc202824565)

[3.2.2 流水线各阶段详细设计 15](#_Toc202824566)

[3.2.3 流水线寄存器设计 15](#_Toc202824567)

[3.3 流水线冒险处理机制 16](#_Toc202824568)

[3.3.1 数据冒险分析与处理 16](#_Toc202824569)

[3.3.2 控制冒险与分支预测 16](#_Toc202824570)

[3.3.3 结构冒险的避免 17](#_Toc202824571)

[4. 设计详解 17](#_Toc202824572)

[4.1 基础功能实现 17](#_Toc202824573)

[4.1.1 Y86指令集完整实现 17](#_Toc202824574)

[4.1.2 五级流水线核心架构 18](#_Toc202824575)

[4.1.3 完整的冒险处理机制 18](#_Toc202824576)

[4.2 RAS预测机制的创新设计 19](#_Toc202824577)

[4.2.1 问题分析与设计动机 19](#_Toc202824578)

[4.2.2 核心设计思想与创新点 19](#_Toc202824579)

[4.2.3 双向移位寄存器设计 20](#_Toc202824580)

[4.2.4 预测执行机制的集成 20](#_Toc202824581)

[4.2.5 预测错误检测与恢复机制 21](#_Toc202824582)

[4.3 硬件平台适配与优化 23](#_Toc202824583)

[4.3.1 周期计数并显示功能 23](#_Toc202824584)

[4.3.2 模式切换功能 24](#_Toc202824585)

[4.3.3 系统周期分频 25](#_Toc202824586)

[5. 实验步骤 25](#_Toc202824587)

[5.1 硬件平台准备 25](#_Toc202824588)

[5.2 周期计数功能实现 25](#_Toc202824589)

[5.3 模式切换功能实现 25](#_Toc202824590)

[6. 仿真调试总结 26](#_Toc202824591)

[6.1 测试程序 26](#_Toc202824592)

[测试用例讲解： 26](#_Toc202824593)

[1. 初始化与栈设置：初始化栈指针，调用主函数，程序结束时停止处理器 26](#_Toc202824594)

[2. main 函数：准备参数并调用求和函数 sum 27](#_Toc202824595)

[3. sum 函数（核心逻辑）:循环累加数组元素的和 27](#_Toc202824596)

[6.2 测试结果 28](#_Toc202824597)

[6.2.1 流水线化 28](#_Toc202824598)

[6.2.2 加载/使用冒险 29](#_Toc202824599)

[6.2.3 分支预测 30](#_Toc202824600)

[6.2.4 返回地址预测 31](#_Toc202824601)

[6.2.5 异常 32](#_Toc202824602)

[6.2.6 硬件结果展示 34](#_Toc202824603)

[7. 实验总结与心得 34](#_Toc202824604)

[7.2 团队学习与成长 35](#_Toc202824605)

[8. 未来优化方向 35](#_Toc202824606)

[8.1 单拍/连续模式切换功能的异步化 35](#_Toc202824607)

[8.2 性能优化方向 36](#_Toc202824608)

[8.3 功能扩展方向 36](#_Toc202824609)

[9. 参考文献 36](#_Toc202824610)

[10. 附录 36](#_Toc202824611)

[10.1 成员心得总结 36](#_Toc202824612)

[10.2 工作日志 39](#_Toc202824613)

[10.3贡献度表 41](#_Toc202824614)

## 1. 需求分析

### 1.1 项目背景与目标

本项目的主要目标是实现一个功能完整、性能优良的Y86流水线处理器。具体包括：实现Y86指令集架构的所有核心指令，确保指令执行的正确性和完整性；采用五级流水线结构，实现指令的并行执行，提高处理器的吞吐量；实现完整的冒险处理机制，包括数据前推、流水线暂停、分支预测等，确保流水线的正确执行；引入创新的RAS预测机制，显著优化`ret`指令的执行性能；在FPGA平台上成功验证设计，实现稳定可靠的硬件系统。

### 1.2 功能需求分析

#### 1.2.1 基础功能需求

指令集支持：处理器必须完整支持Y86指令集架构，包括数据传送指令（irmovq、rrmovq、mrmovq、rmmovq）、算术逻辑运算指令（addq、subq、andq、xorq）、跳转指令（jmp、jle、jl、je、jne、jge、jg）、条件传送指令（cmovle、cmovl、cmove、cmovne、cmovge、cmovg）、栈操作指令（call、ret、pushq、popq）以及控制指令（halt、nop）。

流水线架构：处理器必须采用五级流水线结构，包括取指（Fetch）、译码（Decode）、执行（Execute）、访存（Memory）、写回（Writeback）五个阶段。每个阶段都有独立的功能和相应的流水线寄存器，确保指令能够并行执行。

寄存器文件：实现包含8个64位通用寄存器的寄存器文件，支持同时读取两个操作数和写入一个结果。寄存器文件必须支持前推机制，能够将尚未写回的结果直接传递给需要的指令。

#### 1.2.2 高级功能需求

**冒险处理机制**：实现完整的流水线冒险处理机制，包括数据冒险的检测和处理、控制冒险的预测和恢复、结构冒险的避免等。

**分支预测**：实现基本的分支预测机制，对于条件跳转指令采用静态预测策略。

**RAS预测机制**：这是本项目的核心创新功能，必须实现完整的返回地址栈预测机制。

#### 1.2.3 平台适配需求

FPGA平台支持：处理器设计必须能够在Minisys实验板上成功综合和实现，满足FPGA平台的资源约束和时序要求。

调试支持功能：实现丰富的调试和演示功能，包括LED状态指示、数码管周期显示、执行模式切换等。

### 1.3 非功能需求分析

#### 1.3.1 性能需求

时钟频率：考虑到Minisys平台的限制和教学演示的需要，处理器的目标工作频率设定为50MHz。

流水线效率：理想情况下，五级流水线应该能够达到每个时钟周期完成一条指令的吞吐量，即CPI接近1。

预测准确率：RAS预测机制的准确率目标设定为95%以上。

#### 1.3.2 资源需求

逻辑资源：Minisys实验板搭载的XC7A100T FPGA提供了101,440个逻辑单元（LUT），本项目的目标是将逻辑资源使用率控制在50%以下。

存储资源：FPGA提供了4,976,640位的总RAM资源，项目需要合理分配这些存储资源。

#### 1.3.3 可靠性需求

功能正确性：处理器必须能够正确执行所有Y86指令，通过完整的测试程序验证功能的正确性。

系统稳定性：处理器必须能够在各种工作条件下稳定运行。

### 1.4 成员分工

本项目采用团队协作的方式完成，各成员分工明确，协作高效：

**李思琪**：负责RAS（返回地址栈）预测机制的设计与实现，包括预测算法、错误检测与恢复机制的设计，以及数据冒险逻辑的修正和优化。同时负责技术方案报告的撰写和整体架构的设计。

**陈子容**：负责Y86流水线处理器的核心Verilog代码编写，包括五级流水线各阶段的实现、冒险处理逻辑的编码，以及系统的调试和安全性优化。同时负责实现文档的撰写和代码规范的制定。

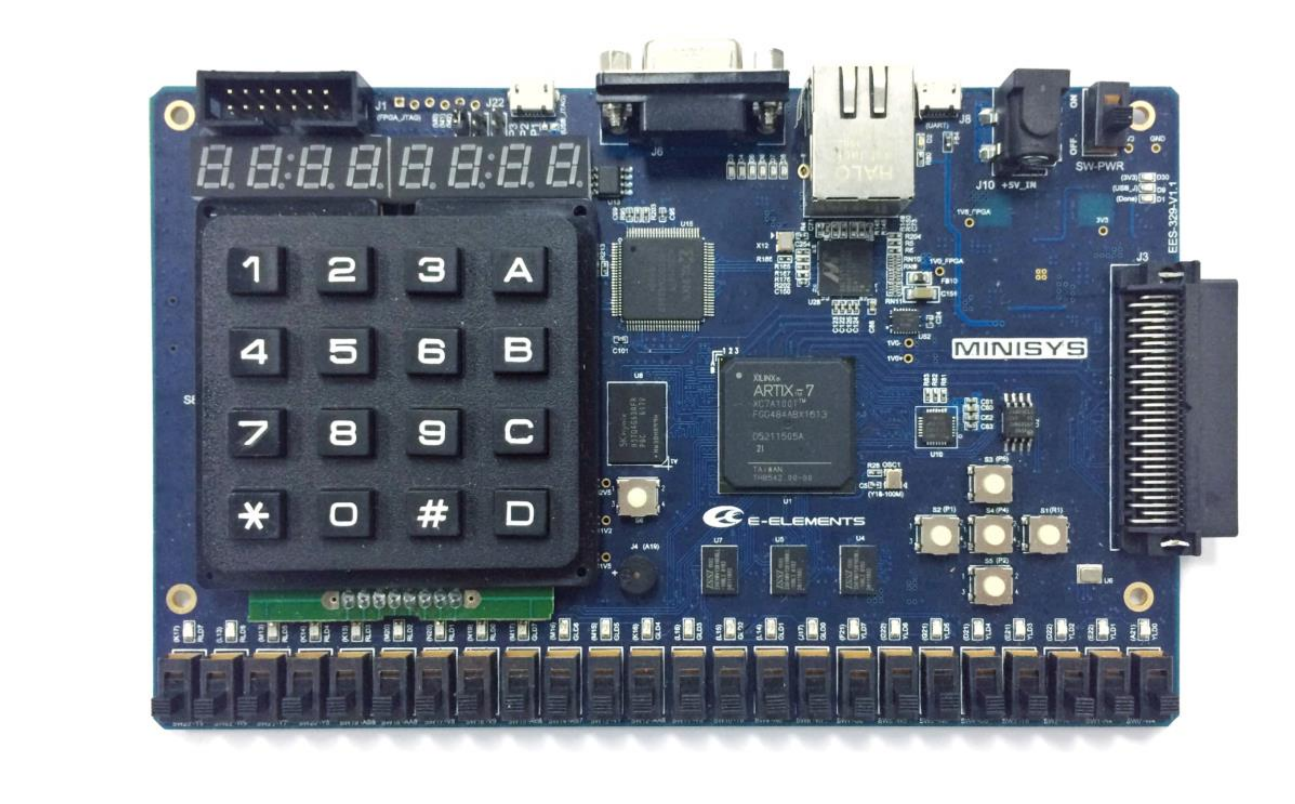
**计子毅**：负责硬件平台的调试和优化，包括Vivado开发环境的配置、实验板管脚规范的研究、模式切换功能的实现，以及系统时钟的优化设计。同时负责硬件部分文档的撰写。

**鹿一凡**：负责硬件调试和用户界面的实现，包括LED显示系统的设计、周期计数数码管显示功能的实现，以及测试用例的硬件验证。同时负责硬件部分文档的撰写。

## 2. 硬件环境

### 2.1 实验板概述

Minisys实验板是专为计算机组成原理课程设计的FPGA开发平台，以Xilinx Artix-7系列FPGA（型号XC7A100T-1FGG484C）为核心处理器件。该FPGA采用28nm工艺制造，提供了丰富的可编程逻辑资源和灵活的配置能力，非常适合实现复杂的数字系统设计。



Minisys实验板整体布局图

XC7A100T FPGA的主要技术参数包括：101,440个逻辑单元（Logic Cells），15,850个可配置逻辑块（CLB），每个CLB包含8个6输入查找表（LUT）和16个触发器；4,860Kb的块RAM资源，可配置为各种位宽和深度的存储器；240个DSP48E1数字信号处理单元，支持高效的乘法和累加运算；6个时钟管理单元（CMT），每个CMT包含1个混合模式时钟管理器（MMCM）和1个锁相环（PLL）。

### 2.2 本实验用到的板卡资源

#### 2.2.1 LEDs（红、黄、绿各8个）

实验板上24个LED灯（红、绿、黄分别8个），板上标号为RLD7~0、GLD7~0和YLD7~0。当FPGA相应管脚的输出为高电平时，所连接的LED灯被点亮，否则灯熄灭。

在我们的处理器设计中，LED系统被用作重要的状态指示和调试工具。红色LED组用于显示处理器的基本运行状态，包括时钟活动、复位状态、异常状态等。绿色LED组用于显示流水线各阶段的活动状态，每个LED对应一个流水线阶段，当该阶段有有效指令时LED点亮。黄色LED组用于显示冒险检测和RAS预测的状态信息，包括数据冒险、控制冒险、预测命中等。

#### 2.2.2 按键开关

板上共有6个按键开关（S1~S6），其中的S6按键被选作FPGA的复位按键。S5用作切换单拍/连续模式，S4用作单拍时的周期切换。

在我们的设计中，按键的功能分配如下：S6用作系统复位，按下时将处理器恢复到初始状态；S5用作执行模式切换，在连续执行模式和单步执行模式之间切换；S4用作单步执行触发，在单步模式下每按一次执行一个时钟周期。

#### 2.2.3 7段数码管

Minisys实验板上有两个4位带小数点的七段数码管，下图显示了它们与主芯片的连接方式。其中A7-A0是数码管8个位的使能信号，而CA-CG/DP则对应各个位上七个段以及小数点的触发信号。需要注意的是，使能信号和触发信号都是低电平触发的。

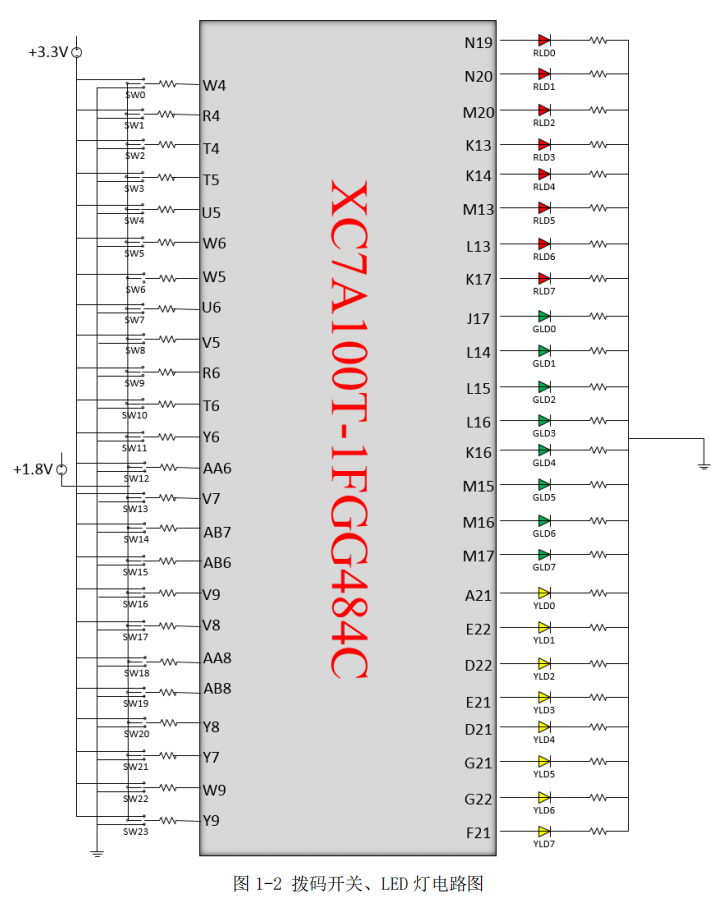


图2.2：数码管与主芯片的连接方式

下图以数码管中最右侧的A0数码管为例说明了Minisys板卡上的7-段数码管的连接方式。8个位中的各个相应的段及小数点分别连接到一组低电平触发的引脚上，他们被称为CA、CB、CC、…、CG、DP，其中，CA接到这8个数码管中每一个数码管A段的负极,CB接到这8个数码管中每一个数码管B段的负极，以此类推。

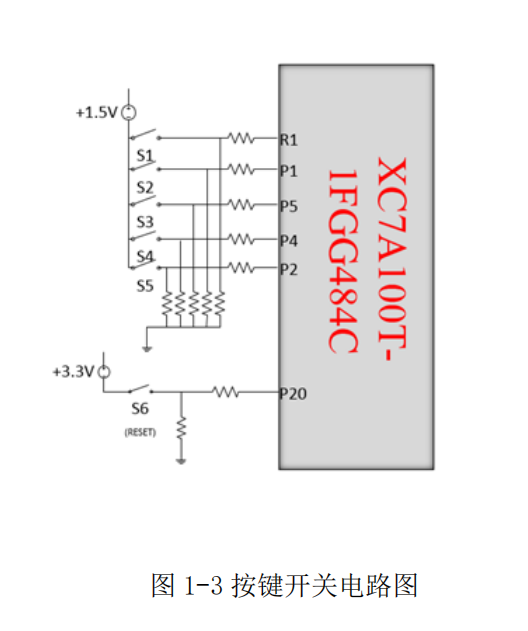


图2.3：数码管段选信号连接示意图

此外，每一个数码管都有一个使能信号A[7:0]。A[7:0]通过一个反相器接到对应数码管的每一个段的正极上。比如说，只有到A[0]为0的时候，最右侧数码管的显示才会受到CA…CG这几个信号的驱动。

要想让每个数码管显示不同的数字，使能信号（A[7:0]）和段信号（CA…CG）必须依次地被持续驱动，数码管之间的刷新速度应该足够快这样就看不出来数码管之间在闪烁。举个例子，如果想在数码管0上显示数字3而数码管1上显示数字9，可以先把CA…CG设置为显示数字3，并拉低A[1]信号，然后再把CA…CG设置为显示数字9并拉高A[1]拉低A[2]。刷新频率可以设置为2ms刷新一次，这样人眼就看不出闪烁了。

在我们的设计中，数码管主要用于显示处理器的执行周期数。通过实时显示周期计数，用户可以直观地观察处理器的执行进度和性能表现。

## 3. 实验原理

### 3.1 Y86指令集体系结构

#### 3.1.1 程序员可见状态

Y86处理器的程序员可见状态是指程序员在编写汇编代码时能够直接访问和操作的硬件资源。这些状态的设计直接影响指令集的表达能力和编程的便利性。

**寄存器文件**是Y86处理器的核心存储资源，包含8个64位通用寄存器。这些寄存器分别命名为%rax、%rcx、%rdx、%rbx、%rsp、%rbp、%rsi、%rdi，基本保持了与x86架构的兼容性。每个寄存器都可以作为算术逻辑运算的操作数，也可以用于地址计算和数据传送。其中%rsp寄存器具有特殊意义，用作栈指针，指向当前栈顶的位置。

**条件码寄存器**包含三个单位标志：零标志（ZF）、符号标志（SF）和溢出标志（OF）。这些标志由算术逻辑运算指令自动设置，并被条件跳转和条件传送指令用于判断执行条件。

**程序计数器（PC）**是一个64位寄存器，存储当前正在执行指令的地址。PC的值在每个指令周期中自动更新，通常递增到下一条指令的地址，但在跳转、调用、返回等控制转移指令执行时会被修改为目标地址。

**内存**被视为一个大的字节数组，支持64位地址空间。Y86采用小端字节序，即多字节数据的低位字节存储在低地址。

**状态码（Stat）**是一个特殊的状态寄存器，描述程序执行的总体状态。状态码包括正常运行（AOK）、遇到halt指令（HLT）、非法指令（INS）、非法地址（ADR）等状态。

#### 3.1.2 指令集设计原理

Y86指令集的设计遵循了RISC（精简指令集计算机）的设计理念，在保持足够表达能力的前提下，尽可能简化指令的格式和语义。

**指令格式的统一性**是Y86设计的重要特点。所有指令都采用变长编码格式，第一字节包含指令码和功能码，后续字节根据指令类型包含寄存器标识符、立即数、地址偏移等信息。

**操作数模式的简化**是另一个重要设计决策。Y86将x86复杂的寻址模式简化为几种基本模式：寄存器模式、立即数模式、基址+偏移模式等。

**指令语义的正交性**确保了不同指令之间的功能独立，避免了复杂的指令交互。

#### 3.1.3 指令分类与编码

Y86指令集可以按照功能分为几个主要类别，每个类别都有特定的编码规则和执行特点。

**数据传送指令**是最基础的指令类别，包括立即数到寄存器传送（irmovq）、寄存器到寄存器传送（rrmovq）、内存到寄存器传送（mrmovq）、寄存器到内存传送（rmmovq）等。

**算术逻辑运算指令**包括加法（addq）、减法（subq）、按位与（andq）、按位异或（xorq）等基本运算。这些指令都采用双操作数格式，第一个操作数既是源操作数也是目标操作数，第二个操作数只作为源操作数。

**控制转移指令**包括无条件跳转（jmp）、条件跳转（jle、jl、je、jne、jge、jg）、过程调用（call）、过程返回（ret）等。

**栈操作指令**包括压栈（pushq）和出栈（popq）操作，这些指令自动维护栈指针（%rsp）的值。

### 3.2 五级流水线架构设计

#### 3.2.1 流水线设计理论基础

流水线技术是现代处理器提高性能的核心技术之一，其基本思想是将指令的执行过程分解为多个独立的阶段，不同的指令可以在不同的阶段并行执行，从而提高指令的吞吐量。

**流水线的理论基础**建立在指令执行过程的可分解性上。传统的顺序执行模式中，每条指令必须完全执行完毕后才能开始下一条指令的执行。而在流水线模式中，当前指令进入下一个执行阶段时，新的指令可以进入当前阶段开始执行。

理想情况下，n级流水线可以将指令的吞吐量提高n倍，但实际情况中由于各种冒险和控制问题的存在，性能提升会有所折扣。

**流水线深度的选择**是设计中的重要决策。更深的流水线可以提高时钟频率，但也会增加冒险的概率和控制的复杂性。五级流水线是经典的设计选择，在性能和复杂度之间取得了良好的平衡。

#### 3.2.2 流水线各阶段详细设计

**取指阶段（Fetch Stage）**是流水线的第一个阶段，负责从指令内存中获取指令并进行初步解析。取指阶段的主要功能包括：根据程序计数器（PC）的值从指令内存中读取指令字节；解析指令的第一字节，提取指令码（icode）和功能码（ifun）；根据指令格式提取寄存器标识符（rA、rB）和立即数（valC）；计算下一条指令的地址（valP）；根据指令类型和预测策略确定预测的下一条指令地址（predPC）。

**译码阶段（Decode Stage）**负责从寄存器文件中读取操作数，并确定指令执行所需的控制信号。译码阶段的主要功能包括：根据指令类型确定源寄存器标识符（srcA、srcB）；从寄存器文件中读取操作数（valA、valB）；确定目标寄存器标识符（dstE、dstM）；生成执行阶段所需的控制信号。

**执行阶段（Execute Stage）**是流水线的计算核心，负责执行各种算术逻辑运算和地址计算。执行阶段的主要功能包括：根据指令类型选择ALU的运算功能；执行算术逻辑运算，产生结果（valE）；计算内存访问的有效地址；设置条件码寄存器；判断条件跳转和条件传送的条件是否成立。

**访存阶段（Memory Stage）**负责与数据内存的交互，包括数据的读取和写入。访存阶段的主要功能包括：根据指令类型确定内存访问的类型（读、写、无访问）；使用执行阶段计算的地址访问数据内存；从内存中读取数据（valM）或将数据写入内存；检查内存地址的有效性，处理地址异常。

**写回阶段（Writeback Stage）**是流水线的最后一个阶段，负责将计算结果写回寄存器文件并更新处理器状态。写回阶段的主要功能包括：将执行阶段的结果（valE）或访存阶段的结果（valM）写入目标寄存器；更新处理器的状态码（Stat）；处理异常和中断；完成指令的执行并释放流水线资源。

#### 3.2.3 流水线寄存器设计

流水线寄存器是连接各个流水线阶段的关键组件，负责在时钟边沿传递数据和控制信号。每个流水线寄存器都包含数据字段和控制字段，以及相应的控制逻辑。

**F寄存器**位于取指阶段之前，主要存储预测的程序计数器值（predPC）。F寄存器的控制逻辑支持正常更新、暂停（stall）两种操作模式。

**D寄存器**位于取指和译码阶段之间，存储指令解析的结果和相关控制信号。D寄存器的字段包括状态码（stat）、指令码（icode）、功能码（ifun）、寄存器标识符（rA、rB）、立即数（valC）、下一指令地址（valP）等。

**E寄存器**位于译码和执行阶段之间，存储操作数和执行控制信号。

**M寄存器**位于执行和访存阶段之间，存储ALU计算结果和访存控制信号。

**W寄存器**位于访存和写回阶段之间，存储最终的执行结果。

### 3.3 流水线冒险处理机制

#### 3.3.1 数据冒险分析与处理

数据冒险是流水线处理器面临的主要挑战之一，当一条指令需要使用前面指令产生的结果，而该结果尚未写回寄存器文件时，就会发生数据冒险。

**读后写（RAW）冒险**是最常见的数据冒险类型。当前指令需要读取某个寄存器的值，而前面的指令正在修改该寄存器时，就会发生RAW冒险。

**数据前推（Forwarding）机制**是解决大部分数据冒险的有效方法。前推机制的基本思想是将尚未写回寄存器文件的计算结果直接传递给需要该数据的指令，避免等待写回操作的完成。

**加载/使用冒险**是一种特殊的数据冒险，当一条指令需要使用前一条内存读取指令的结果时发生。由于内存读取的结果要到访存阶段才能获得，无法通过简单的前推机制解决。

#### 3.3.2 控制冒险与分支预测

控制冒险发生在程序的控制流发生改变时，如跳转、调用、返回等指令的执行。由于这些指令的目标地址需要在执行阶段才能确定，而此时流水线已经取指了后续的指令，可能导致取指错误的指令。

**静态分支预测**是本设计采用的基本预测策略。对于条件跳转指令，我们总是预测选择分支，即预测跳转会发生。

**分支预测错误的恢复**是控制冒险处理的关键环节。当条件跳转指令在执行阶段确定实际的跳转方向时，如果与预测不符，就需要清除流水线中错误取指的指令，并从正确的地址重新开始取指。

#### 3.3.3 结构冒险的避免

结构冒险发生在多条指令同时竞争同一硬件资源时。在我们的设计中，通过合理的硬件资源配置和访问调度，基本避免了结构冒险的发生。

**寄存器文件的多端口设计**是避免结构冒险的重要措施。寄存器文件支持同时读取两个操作数和写入一个结果，满足了流水线的并发访问需求。

**指令内存和数据内存的分离**避免了指令获取和数据访问之间的冲突。

## 4. 设计详解

### 4.1 基础功能实现

#### 4.1.1 Y86指令集完整实现

本项目实现了完整的Y86指令集，包括所有基础指令和高级功能。指令集的实现严格遵循Y86规范，确保与标准Y86架构的完全兼容。

**数据传送指令的实现**涵盖了所有数据移动模式。立即数传送指令（irmovq）将64位立即数直接加载到目标寄存器；寄存器传送指令（rrmovq）实现寄存器间的数据复制；内存读取指令（mrmovq）从指定内存地址读取数据到寄存器；内存写入指令（rmmovq）将寄存器数据写入指定内存地址。所有传送指令都支持条件执行，根据当前条件码的状态决定是否执行传送操作。

**算术逻辑运算指令的实现**包括四种基本运算：加法（addq）、减法（subq）、按位与（andq）、按位异或（xorq）。这些指令采用双操作数格式，第一个操作数既是源也是目标，第二个操作数只作为源。运算结果会自动更新条件码寄存器，为后续的条件执行提供判断依据。ALU的设计支持64位运算，能够正确处理有符号和无符号数的运算，包括溢出检测和符号扩展。

**控制转移指令的实现**是流水线设计的重点和难点。无条件跳转指令（jmp）直接修改程序计数器到目标地址；条件跳转指令（jle、jl、je、jne、jge、jg）根据条件码的状态决定是否跳转；过程调用指令（call）将返回地址压栈并跳转到目标地址；过程返回指令（ret）从栈中弹出返回地址并跳转。控制转移指令的实现需要与分支预测和RAS预测机制紧密配合，确保预测的准确性和错误恢复的正确性。

**栈操作指令的实现**自动维护栈指针的状态。压栈指令（pushq）将寄存器值写入栈顶并递减栈指针；出栈指令（popq）从栈顶读取值到寄存器并递增栈指针。栈操作的实现需要考虑栈的增长方向（向低地址增长）和栈指针的更新时序，确保栈操作的原子性和一致性。

#### 4.1.2 五级流水线核心架构

五级流水线的实现是本项目的核心技术挑战，需要精确的时序控制和复杂的数据路径设计。

**取指阶段的实现**集成了指令内存接口、指令解析逻辑、PC管理和RAS预测机制。指令内存采用同步读取设计，在时钟上升沿根据PC值读取指令。指令解析逻辑能够在一个时钟周期内完成指令的完整解析，包括指令码提取、寄存器标识符解析、立即数提取等。PC管理逻辑根据指令类型和预测结果选择下一个PC值，支持顺序执行、分支跳转、函数调用等多种控制流模式。

**译码阶段的实现**包括寄存器文件访问、操作数前推、控制信号生成等功能。寄存器文件采用三端口设计，支持同时读取两个操作数和写入一个结果。操作数前推逻辑实现了从执行、访存、写回阶段到译码阶段的数据前推，能够处理各种数据依赖情况。控制信号生成逻辑根据指令类型产生后续阶段所需的控制信号，包括ALU功能选择、内存访问类型、寄存器写入使能等。

**执行阶段的实现**以ALU为核心，集成了条件码生成、地址计算、条件判断等功能。ALU支持Y86指令集的所有运算类型，采用64位数据通路，能够在一个时钟周期内完成运算。条件码生成逻辑根据ALU运算结果设置零标志、符号标志、溢出标志。地址计算逻辑为内存访问指令计算有效地址。条件判断逻辑为条件跳转和条件传送指令评估执行条件。

**访存阶段的实现**负责数据内存的访问和地址有效性检查。数据内存采用同步访问设计，支持读写操作。地址有效性检查能够检测非法内存访问，防止程序访问未定义的内存区域。访存阶段还集成了RAS预测错误检测逻辑，通过比较预测地址和实际地址来判断预测的正确性。

**写回阶段的实现**完成指令执行的最后步骤，包括寄存器写入、状态更新、异常处理等。寄存器写入逻辑将执行结果或内存读取结果写入目标寄存器。状态更新逻辑维护处理器的运行状态，包括正常运行、停机、异常等状态。异常处理逻辑能够处理各种异常情况，确保系统的稳定性。

#### 4.1.3 完整的冒险处理机制

冒险处理机制是流水线正确执行的关键保障，本项目实现了完整的数据冒险、控制冒险、结构冒险处理机制。

**数据前推机制的实现**能够处理大多数数据冒险情况。前推逻辑在译码阶段比较当前指令的源寄存器与流水线中所有后续指令的目标寄存器，当发现匹配时选择最新的结果作为操作数。前推的数据源包括执行阶段的ALU结果、访存阶段的ALU结果和内存读取结果、写回阶段的最终结果。前推逻辑的优先级严格按照流水线阶段的新旧程度确定，确保总是使用最新的数据。

**流水线暂停机制的实现**用于处理无法通过前推解决的加载/使用冒险。当检测到当前指令需要使用前一条内存读取指令的结果时，暂停逻辑会阻止取指和译码阶段的进行，同时在执行阶段插入一个气泡（nop指令）。暂停机制的实现需要精确的时序控制，确保暂停和恢复的正确性。

**分支预测机制的实现**采用静态预测策略，对条件跳转指令总是预测选择分支。预测逻辑在取指阶段根据指令类型选择预测的下一个PC值。当分支预测错误时，恢复逻辑会清除流水线中错误取指的指令，并从正确的地址重新开始取指。分支预测的实现需要与流水线控制逻辑紧密配合，确保预测错误时的正确恢复。

### 4.2 RAS预测机制的创新设计

#### 4.2.1 问题分析与设计动机

在深入分析传统Y86流水线处理器的性能瓶颈时，我们发现`ret`指令的执行效率是一个关键的限制因素。传统设计中，`ret`指令需要经历完整的流水线流程才能获得返回地址，具体表现为以下3个方面的性能损失：

**执行延迟问题**：`ret`指令在取指阶段被识别后，需要等待到访存阶段才能从栈内存中读取返回地址。这意味着从指令被取指到获得目标地址，需要经过译码、执行、访存三个阶段，总计3个时钟周期的延迟。在这3个周期中，流水线无法取指新的指令，导致流水线气泡的产生。

**内存访问开销**：每次`ret`指令的执行都需要访问数据内存来读取返回地址。内存访问相对于寄存器访问是一个较慢的操作，特别是在存在内存层次结构的系统中，内存访问可能需要多个时钟周期才能完成。

**流水线效率影响**：由于返回地址获取的延迟，后续指令的取指过程被迫暂停，这不仅降低了指令的吞吐量，还影响了流水线的整体效率。在函数调用密集的程序中，这种影响会被放大，显著降低程序的执行性能。

#### 4.2.2 核心设计思想与创新点

基于对问题的深入分析，我们提出了返回地址栈（RAS）预测机制的创新设计。该设计的核心思想建立在以下几个关键观察和洞察之上：

**指令配对特性的利用**：在结构化的程序中，`call`和`ret`指令严格遵循配对原则。每个函数调用（`call`）都有对应的函数返回（`ret`），这种配对关系为预测提供了强有力的依据。通过跟踪`call`指令压入的返回地址，我们可以预测对应`ret`指令的目标地址。

**栈式执行模式的映射**：函数调用和返回的执行模式天然地遵循后进先出（LIFO）的栈特性。最近调用的函数最先返回，这种执行模式可以直接映射到硬件栈结构上。通过在硬件中维护一个返回地址栈，我们可以准确地跟踪函数调用的嵌套关系。

**零延迟预测的实现**：通过将RAS集成到取指阶段，我们可以在识别`ret`指令的同时立即获得预测的返回地址。这种设计将返回地址的获取延迟从3个周期降低到0周期，实现了真正的零延迟预测。

#### 4.2.3 双向移位寄存器设计

在RAS的具体实现中，我们设计并比较了两种不同的硬件架构方案，最终选择了性能更优的双向移位寄存器架构。

传统寄存器堆方案的局限性：传统的RAS实现通常采用寄存器堆加栈指针的设计。这种方案使用一个深度为16的寄存器堆存储返回地址，通过4位栈指针管理栈顶位置。栈指针的译码需要4-16译码器，在FPGA实现中通常需要多级译码结构（如3-8译码器、2-4译码器的级联）。

多级译码结构的主要问题是组合逻辑延迟较大。在高频率工作的处理器中，译码延迟可能成为关键路径的瓶颈，限制了系统的最高工作频率。此外，多级译码还会增加功耗和硬件复杂度。

**双向移位寄存器方案的优势**：我们采用的双向移位寄存器架构通过移位操作实现栈的压入和弹出功能。该架构的核心特点包括：

硬件结构简化：16级移位寄存器链直接存储返回地址，不需要额外的地址译码逻辑。每级寄存器直接连接到相邻级，形成简单的链式结构。

控制逻辑优化：压栈和弹栈操作通过独立的控制信号直接驱动移位操作。控制逻辑只需要简单的1-2译码，大大降低了组合逻辑的复杂度和延迟。

#### 4.2.4 预测执行机制的集成

RAS预测机制与流水线的集成是设计的关键环节，需要确保预测逻辑与现有的流水线控制逻辑协调工作。

RAS预测逻辑完全集成在取指(Fetch) 阶段，与指令解析逻辑并行工作。

当取指逻辑识别出`call`指令时，即icode==icall,则将计算返回地址（PC + 指令长度）压入栈中。

当识别出`ret`指令时，访问RAS栈顶值，作为预测的返回地址。

#### 4.2.5 预测错误检测与恢复机制

虽然RAS预测在大多数情况下都是准确的，但在某些特殊情况下(如栈溢出）可能出现预测错误。所以需要设计完善的错误检测和恢复机制是确保系统正确性的关键。

**预测错误的产生原因（部分）**：

栈溢出情况：当函数调用深度超过RAS容量（16级）时，最旧的返回地址会被新的地址覆盖，导致深层函数返回时预测错误。

非结构化控制流：程序中存在的非标准控制流转移（如setjmp/longjmp、异常处理、信号处理等）可能破坏call和ret指令的严格配对关系。

**错误检测的实现总思路**：

预测错误的检测在`ret`指令到达访存阶段时进行。此时，访存逻辑从内存中读取真实的返回地址（m\_valM），同时预测值通过流水线寄存器传递到访存阶段（M\_valC）。

**预测值的流水线传递策略**：

为了在访存阶段进行预测错误检测，需要将RAS预测值一直保存直到访存阶段。简单的实现方式时另开一个空间，但是为了优化，充分利用已有的硬件资源，我们采用了资源复用的策略，充分利用流水线寄存器中未使用的字段：

D阶段：`ret`指令的valC字段在原始设计中未使用，我们将预测值存储在D\_valC中。

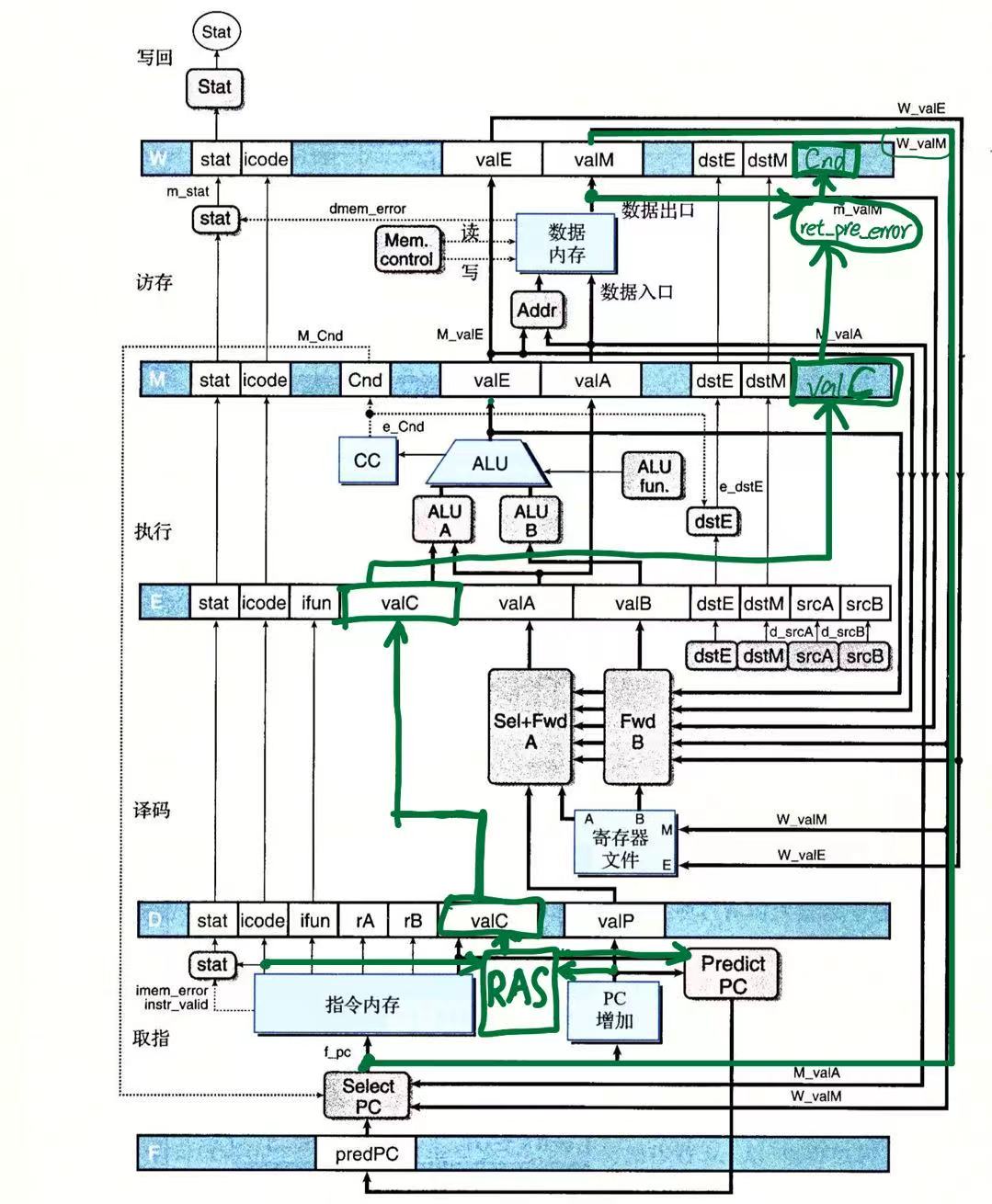
E阶段：预测值随ret指令流水前推到执行阶段，存储在原本的空闲E\_valC中。

M阶段：由于valA和valE字段已被用于栈指针操作，我们新增M\_valC寄存器专门存储预测值。

这种资源复用策略在不增加显著硬件开销的前提下，实现了预测值的保存，且充分利用流水线思想，自然的管理了ret指令及其预测值的对应关系。

**错误恢复机制的实现**：

当M阶段检测到预测错误时，我们将判断信号return\_pred\_err置1，且学习跳转分支预测的思想，将其保存置(新开）W\_Cnd中。下一个周期即是恢复正确指令的周期，通过判断W\_Cod值发现预测错误，则将真实返回地址(此时的w\_valm)，推至Fetch的f\_pc实现执行正确指令。同时进行流水线清理：清除基于错误预测取指的指令，在译码、执行、访存阶段插入气泡。



增加RAS预测机制的流水线处理器示意图

上图展示了集成RAS预测机制的完整Y86流水线处理器架构。图中可以清楚地看到：

1. **取指阶段的RAS集成**：RAS模块位于取指阶段，以icode作为信号，与PC选择逻辑紧密集成，栈顶预测值是存入D\_valC

2. **预测值的流水线传递**：预测值通过valC字段，随ret指令，在流水线中传递

4. **错误检测与恢复**：在访存M阶段进行预测错误检测，通过ret\_pre\_error信号触发恢复机制

### 4.3 硬件平台适配与优化

#### 4.3.1 周期计数并显示功能

为了更直观地展示Y86处理器当前的执行周期数，本系统在硬件中新增了周期计数并显示功能。我们在顶层模块中加入了周期计数器，每当处理器的时钟信号clk\_slow上升沿时，周期计数器会自动加一，用于记录当前执行到的周期数。当程序停机后将保持不变，按下复位键后可重新启动程序。

此外，为了将周期数显示到数码管上，我们使用了数码管扫描显示技术：首先将周期数拆分为十位数和个位数，然后在数码管上交替显示。利用scan\_divider模块产生的扫描时钟clk\_scan周期性切换扫描段，分别显示十位和个位：在scan\_counter == 0时，数码管显示十位数，只有第一位数码管被使能。在scan\_counter == 1时，数码管显示个位数，第二位数码管被使能。其他数码管全部关闭。

每次切换时，通过segment\_decoder模块将4位BCD数字转换为七段数码管的段选信号，最终实现周期计数的数码管显示，便于观察处理器执行过程。

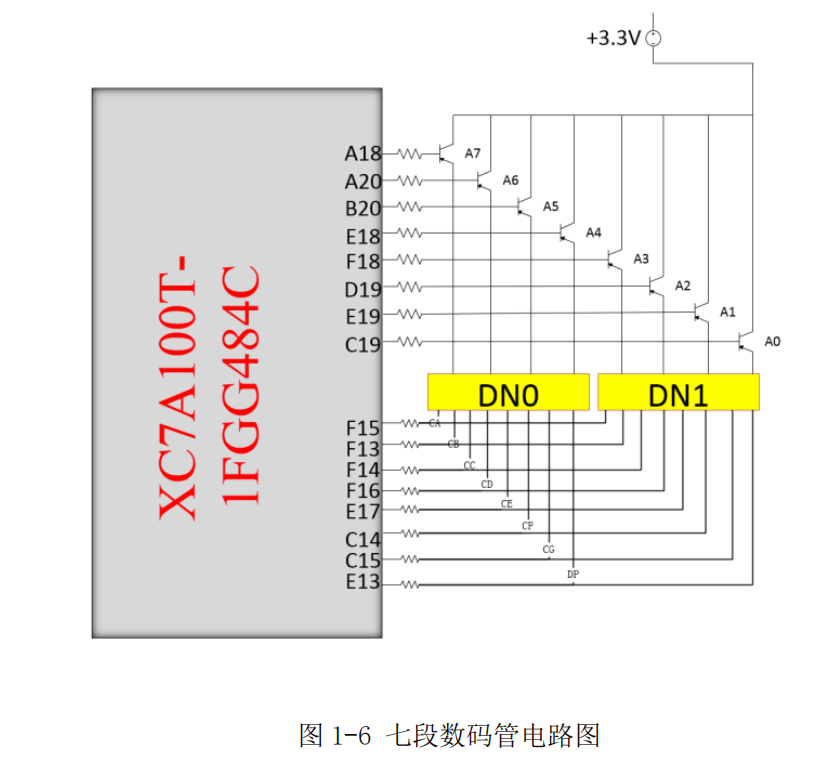


图4.2：数码管周期计数显示效果

#### 4.3.2 模式切换功能

为了方便调试与演示，本系统特别增加了自动/单步模式切换功能，通过按键mode\_btn实现：按下mode\_btn，系统将在自动执行模式与单步执行模式之间切换。

切换通过寄存器step\_mode实现，每次mode\_btn按下后翻转step\_mode状态。

两种模式说明：

**连续模式（step\_mode = 0）**：

时钟分频器clock\_divider正常工作，周期性自动产生clk\_out时钟信号，处理器按设定频率自动运行。

**单拍模式（step\_mode = 1）**：

系统仅在step按钮按下时，才允许时钟分频计数，达到特定条件后产生一次clk\_out，实现手动单步执行，便于精准观察处理器每一步操作。

此模式切换机制大幅提升了调试效率，允许开发者在自动运行与手动单步执行之间灵活切换，观察Y86处理器运行细节。

#### 4.3.3 系统周期分频

由于Minisys板卡包括了一个连接在主芯片Y18管脚的100MHz的晶振，时钟频率过高，不便于观察，因此本系统通过时钟分频模块clock\_divider将100MHz的输入时钟clk\_in分频至较低频率，便于显示与观察：

在连续模式下，时钟按固定分频比（约2Hz）自动输出，处理器周期性运行；

在单拍模式下，只有在step按钮按下时才计数，当计数累积达到指定阈值后输出一个时钟脉冲，手动控制处理器执行。

此外，数码管扫描也使用了专门的scan\_divider模块对时钟进行分频，使数码管显示在稳定亮度与刷新率下正常工作，避免闪烁。

通过上述多层次时钟分频设计，整个系统兼具自动运行、手动单步、稳定显示等多重功能，不仅提高了用户体验，同时也方便调试分析。

## 5. 实验步骤

### 5.1 硬件平台准备

将Y86代码与minisys硬件实验板适配并对接，并进行结果观察，其中代码结果通过RGB小灯进行显示。

### 5.2 周期计数功能实现

编写周期计数功能，并与硬件实验板LED显示板对接，将周期计数器显示出来。

### 5.3 模式切换功能实现

编写单拍和连续执行模式的切换功能，并对接到硬件板上。

## 6. 仿真调试总结

### 6.1 测试程序

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | irmovq stack,%rsp      call main      halt    main:      irmovq array,%rdi      irmovq $4,%rsi      irmovq $8,%r8      call sum      ret    sum:      irmovq $8,%r8      irmovq $1,%r9      xorq %rax,%rax      andq %rsi,%rsi      jmp test    loop:      mrmovq (%rdi),%r10      addq %r10,%rax      addq %r8,%rdi      subq %r9,%rsi    test:      jne loop      ret    stack: |

### 测试用例讲解：

#### ****初始化与栈设置：****初始化栈指针，调用主函数，程序结束时停止处理器

|  |  |
| --- | --- |
| 1  2  3 | irmovq stack, %rsp  ; 将栈指针 %rsp 设置为 stack 标签的地址（栈起始位置）  call main           ; 调用 main 函数（将返回地址压栈）  halt                ; 停止处理器执行（main 返回后执行） |

**关键点**：**stack** 是栈空间的起始地址（由汇编器决定具体地址），**%rsp** 是栈指针寄存器。

#### ****main 函数：****准备参数并调用求和函数 sum

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | main:      irmovq array, %rdi   ; %rdi = 数组起始地址（这里设为 0，可能需在测试时初始化内存）      irmovq $4, %rsi   ; %rsi = 数组元素数量（4 个元素）      irmovq $8, %r8    ; %r8  = 步长值（每个元素占 8 字节）      call sum          ; 调用 sum 函数（返回地址压栈）      ret               ; 返回（返回到 halt） |

寄存器用途：

%rdi：数组起始地址（需在测试时初始化内存）

%rsi：元素计数器（初始为 4）

%r8：步长（8 字节/元素）

#### ****sum 函数（核心逻辑）:循环累加数组元素的和****

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | sum:      irmovq $8, %r8    ; %r8  = 8（步长，冗余操作，main 已设置）      irmovq $1, %r9    ; %r9  = 1（计数器递减步长）      xorq %rax, %rax   ; %rax = 0（清空累加器，存储和）      andq %rsi, %rsi   ; 检查 %rsi 是否为 0（设置条件码 ZF）      jmp test          ; 跳转到 test 标签    loop:      mrmovq 0(%rdi), %r10 ; 从内存 [%rdi] 加载元素到 %r10      addq %r10, %rax      ; 累加：%rax += %r10      addq %r8, %rdi       ; 移动指针：%rdi += 8（指向下一个元素）      subq %r9, %rsi       ; 计数器减 1：%rsi -= 1（设置 ZF 标志）    test:      jne loop  ; 如果 %rsi != 0（ZF=0），跳回 loop      ret       ; 返回（返回到 main） |

寄存器用途：

%rax：累加和（最终结果）

%r10：临时存储加载的元素值

%r9：计数器递减步长（固定为 1）

循环逻辑：

①加载当前元素（mrmovq）

②累加到 %rax（addq）

③指针移动 8 字节（addq %r8, %rdi）

④计数器减 1（subq %r9, %rsi）

⑤如果计数器 %rsi != 0，继续循环（jne loop）

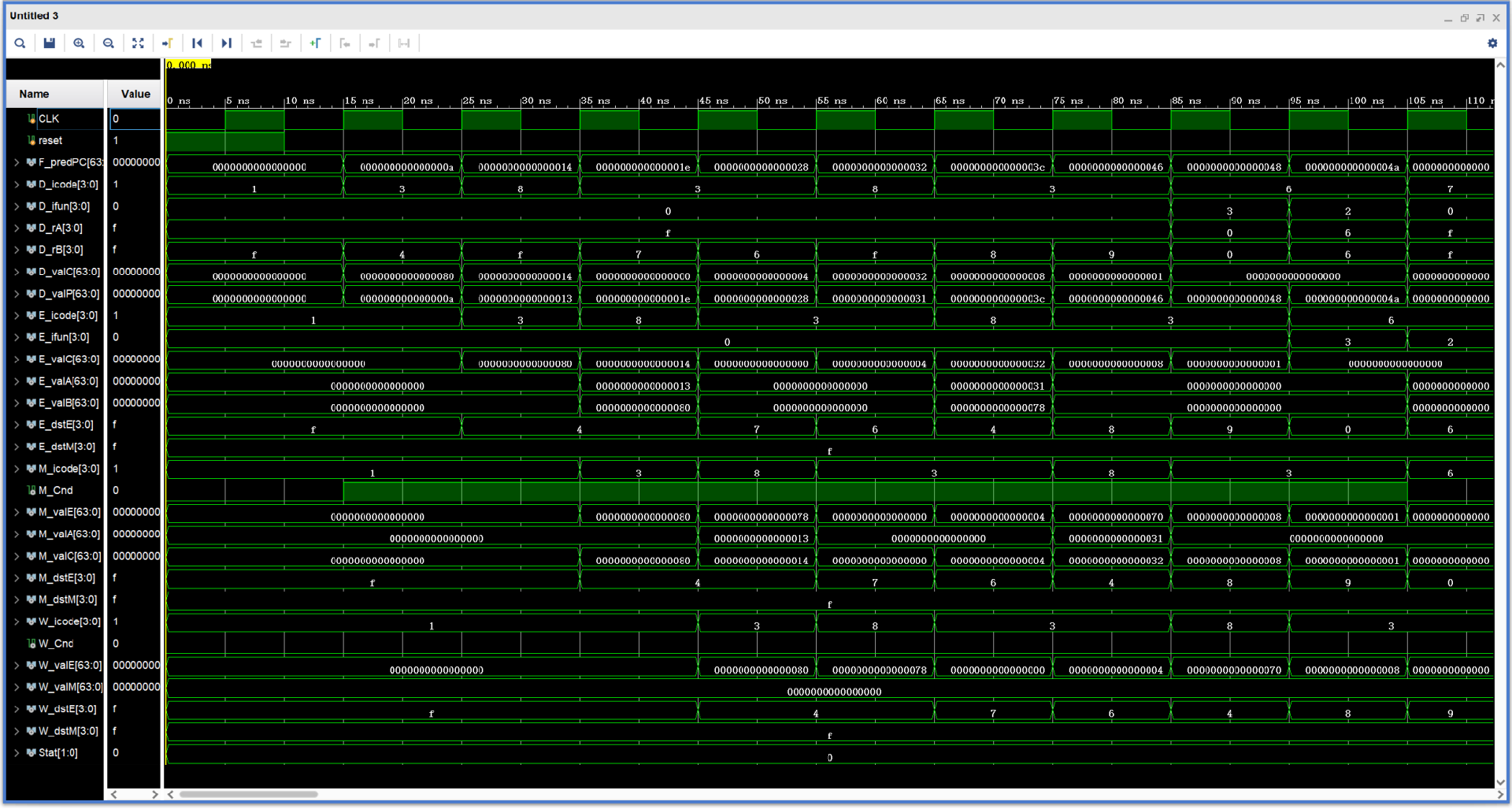
1. 栈空间定义：定义栈的起始位置，供 %rsp 使用

|  |  |
| --- | --- |
| 1 | stack:  ; 栈起始地址（由汇编器分配具体位置） |

### 6.2 测试结果

#### 6.2.1 流水线化

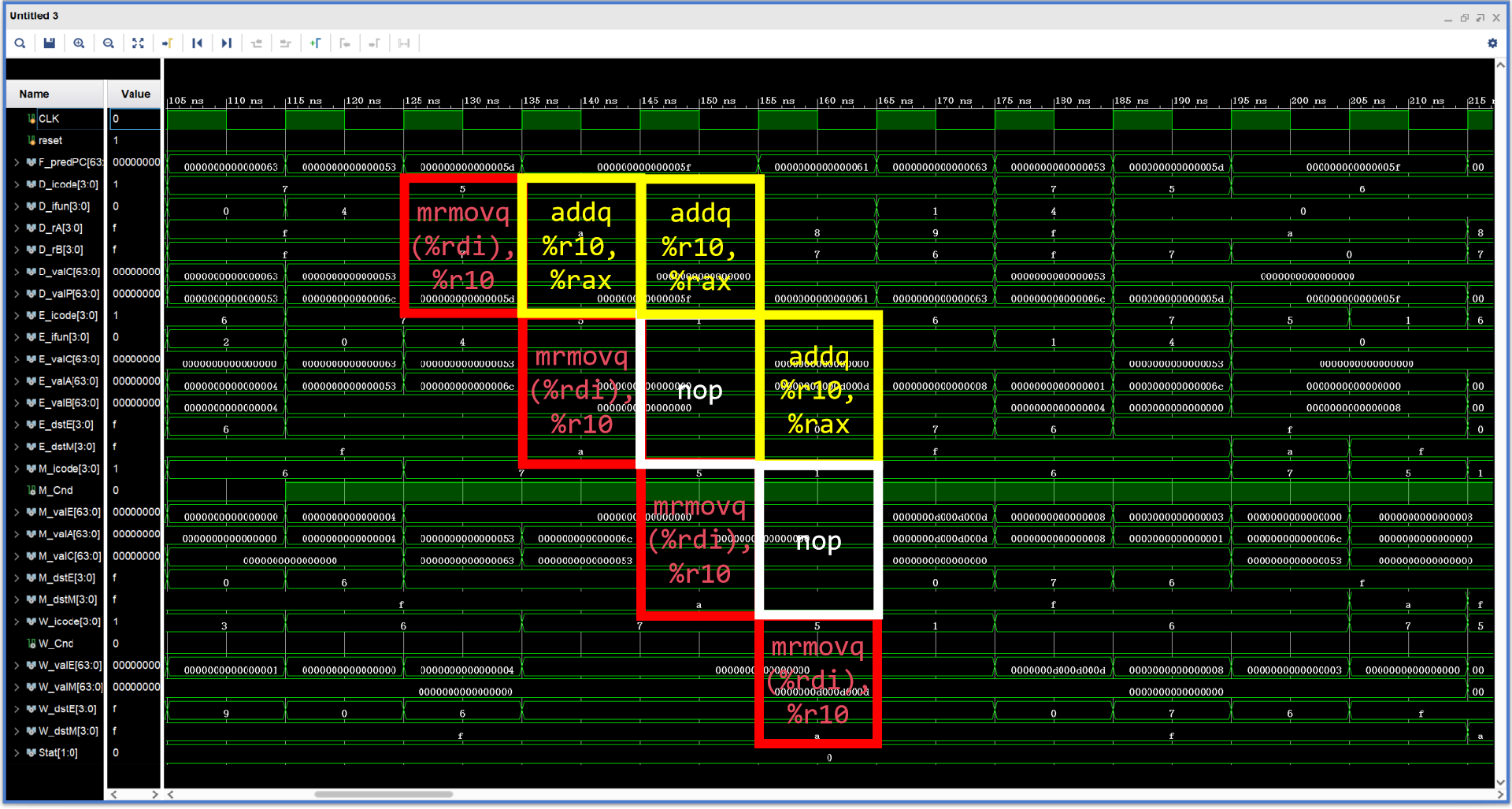
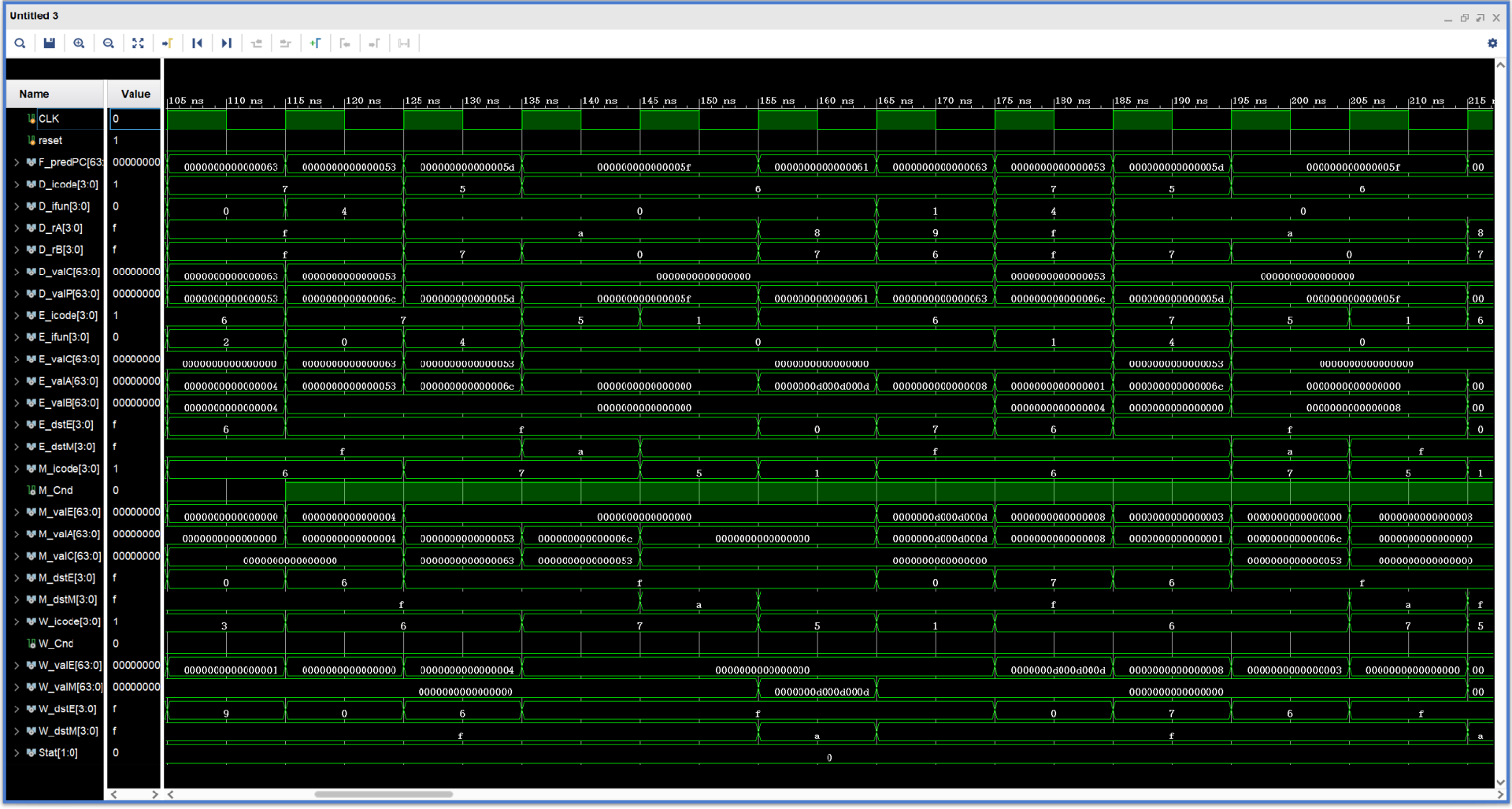
测试程序仿真波形图如下



可以看到波形图清晰地呈现了流水线式的指令处理过程。

#### 6.2.2 加载/使用冒险

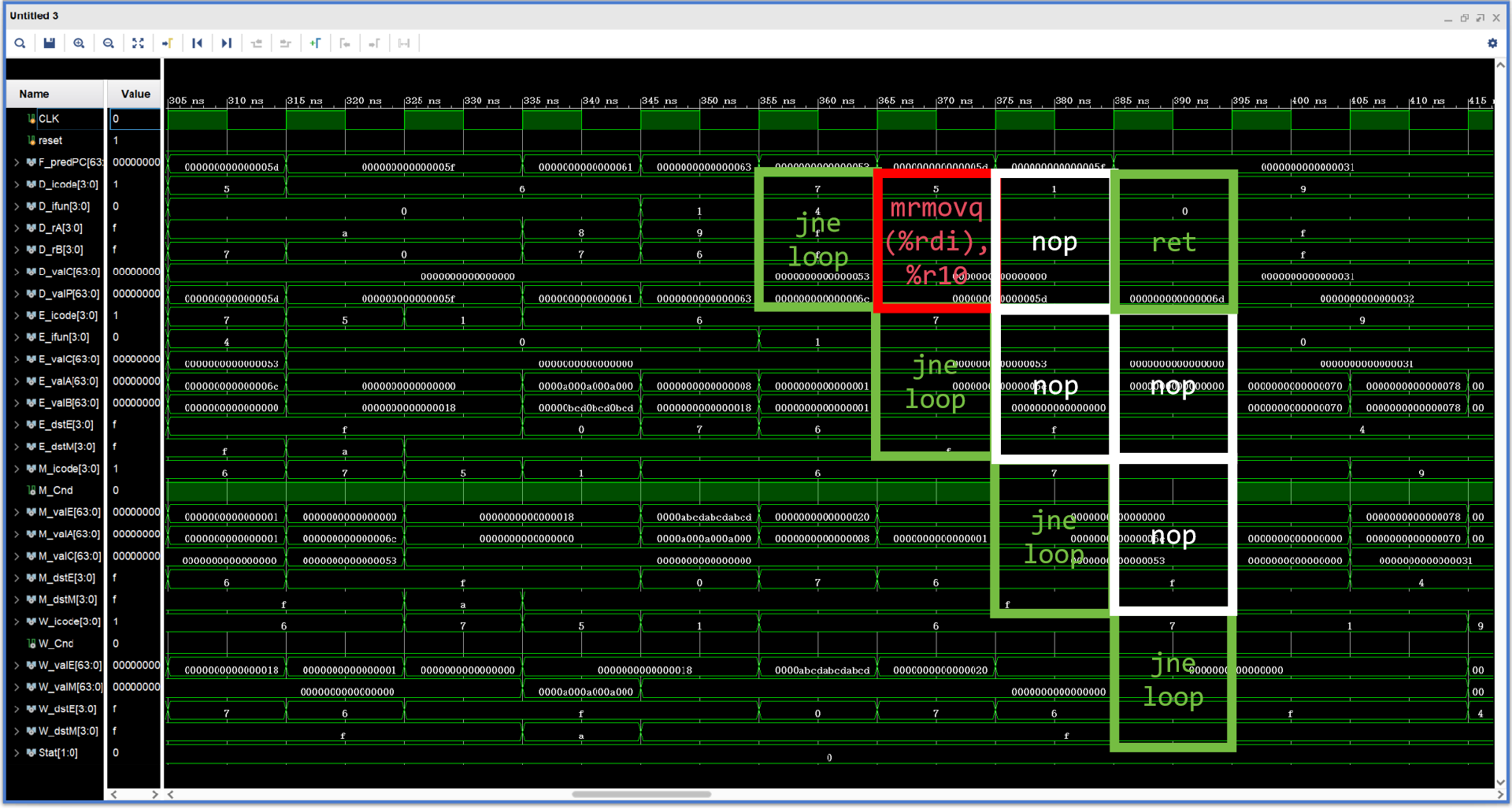
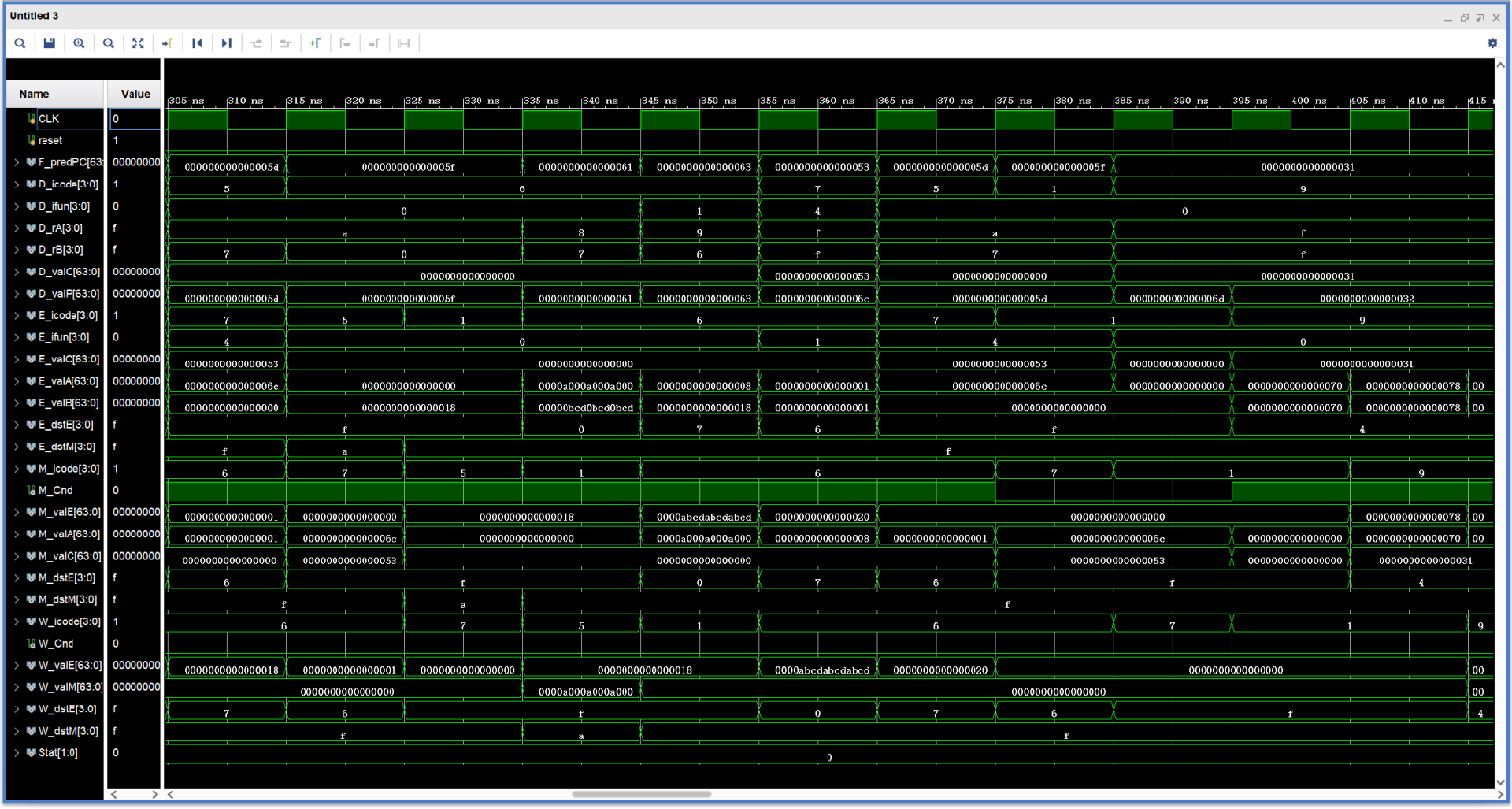
观察如下波形



可以看到，135ns~145ns发生了加载/使用冒险。处理器检测到加载/使用冒险后，在下一周期往E寄存器插入了气泡，并暂停F寄存器与D寄存器，这与流水线控制逻辑对加载/使用冒险的预期处理一致。同时可以看到再下一周期，运算指令得到了正确的操作数0x0000000d000d000d，这验证了数据转发逻辑的正确性。

#### 6.2.3 分支预测

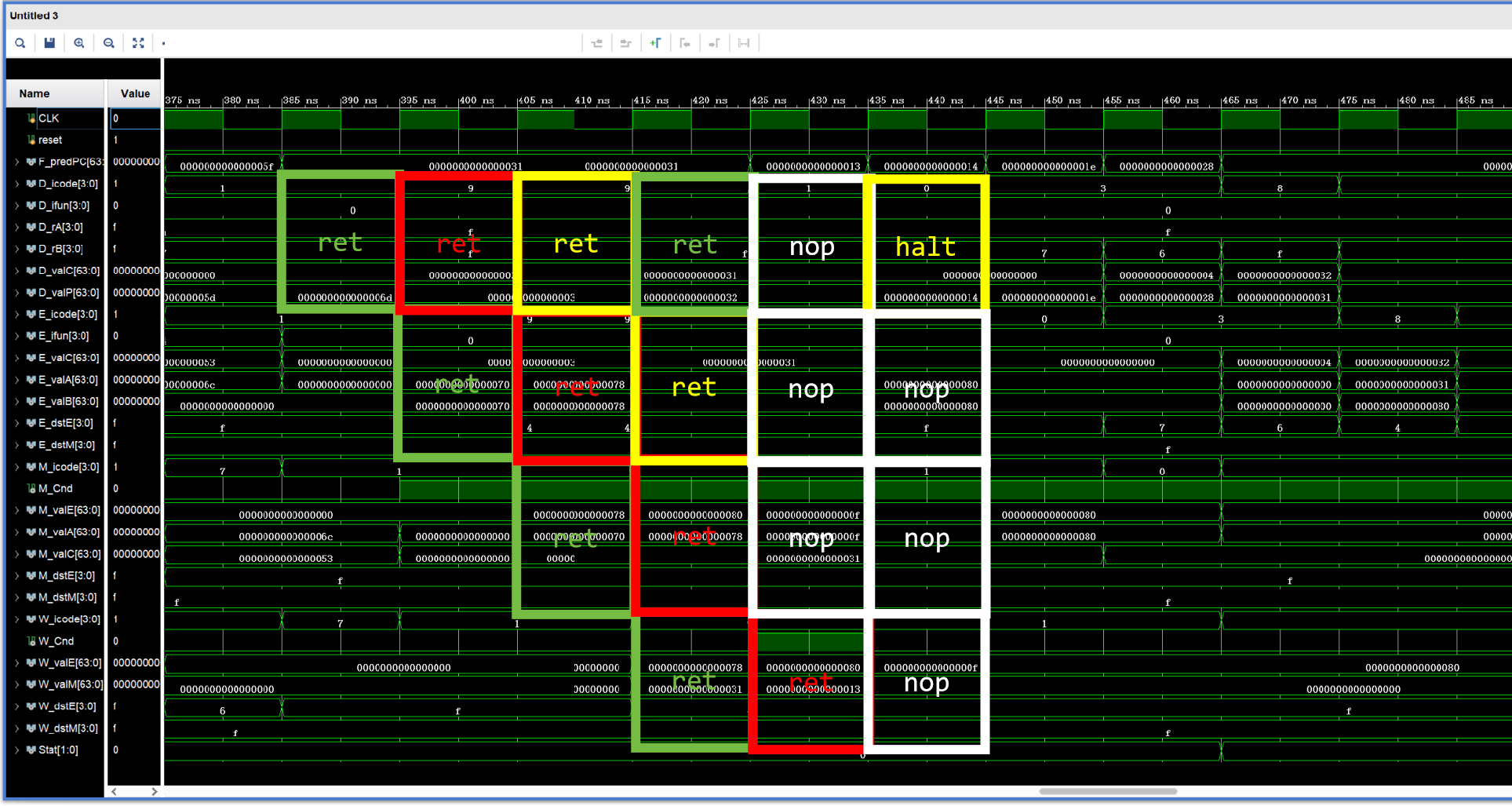
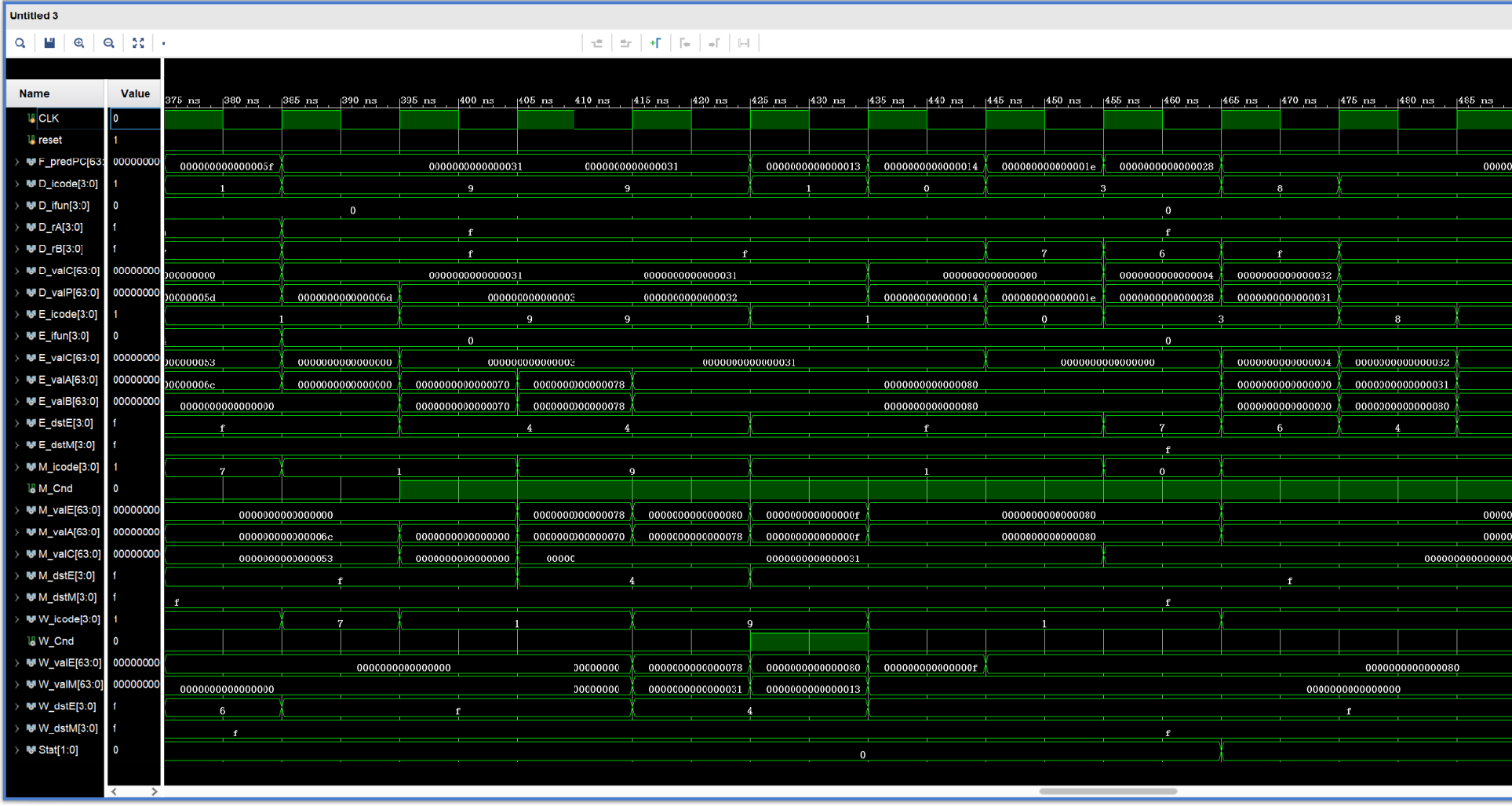
观察如下波形



可以看到，345ns~355ns取出了分支指令，下一周期取出的指令为loop处指令，这与我们期望的预测选择分支策略一致。同时，注意到375ns~385ns处M\_Cnd为0，而上一周期处于执行阶段的指令为分支指令，这说明上一周期发生了分支预测错误，可以看到处理器往D寄存器和E寄存器插入了气泡，这与流水线控制逻辑对分支预测错误的预期处理一致。同时可以看到处理器取出了下一条指令，这验证了分支预测错误恢复逻辑的正确性。

#### 6.2.4 返回地址预测

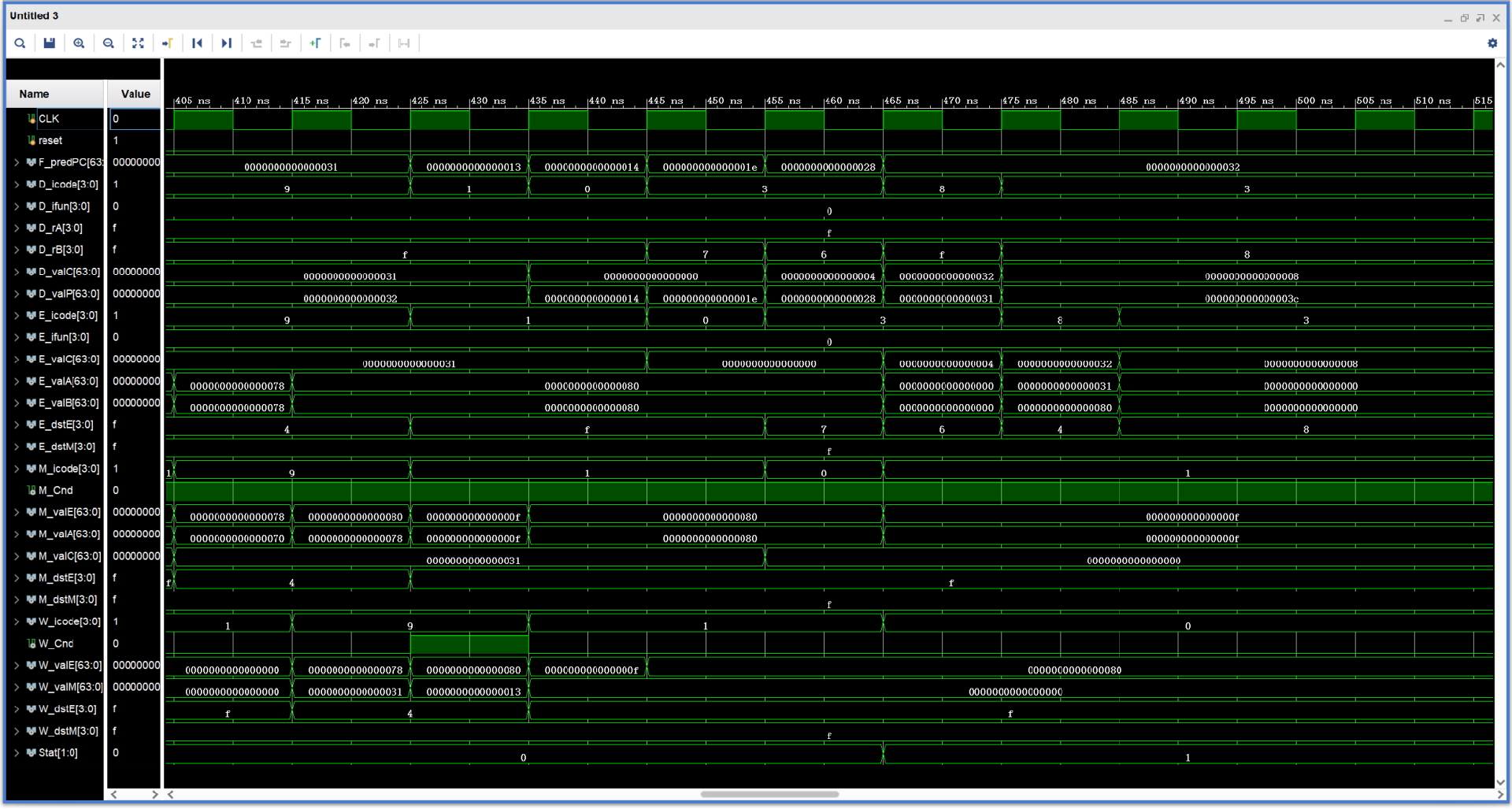
观察如下波形



可以看到，375ns~385ns取出了返回指令，下一周期取出的指令为返回地址处指令，这验证了RAS的正确性。同时，注意到425ns~435ns处W\_Cnd为1，而上一周期处于访存阶段的指令为返回指令，这说明上一周期发生了返回地址预测错误，可以看到处理器往D寄存器，E寄存器和M寄存器插入了气泡，这与流水线控制逻辑对返回地址预测错误的预期处理一致。同时可以看到处理器取出了返回地址处指令，这验证了返回地址预测错误恢复逻辑的正确性。

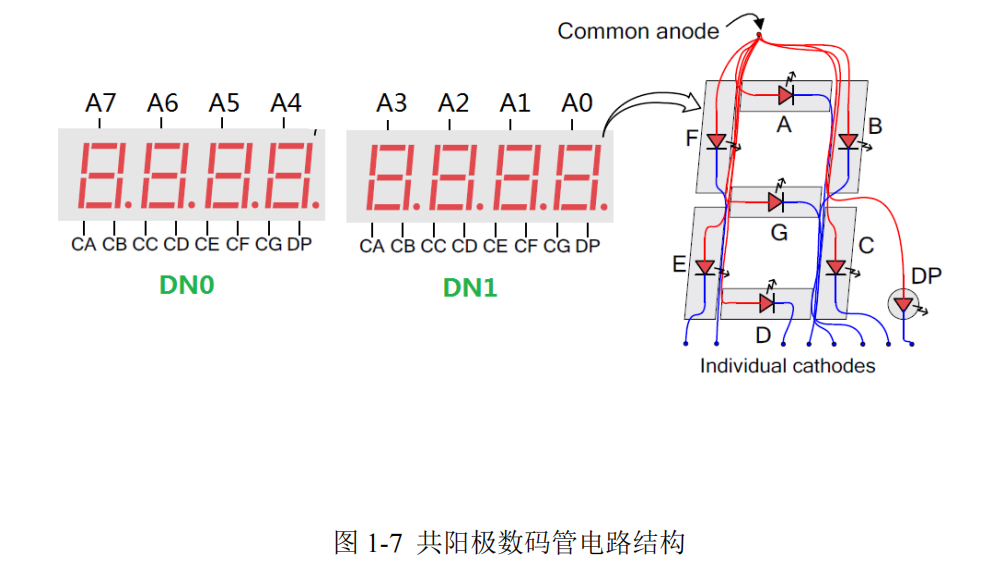
#### 6.2.5 异常

观察如下波形



可以看到，425ns~435ns取出了停机指令，这会引发异常HLT。但由于该指令可能是因预测错误而取出的指令，故处理器不会立刻停机，这符合预期。同时，455ns~465ns处异常指令进入访存阶段，这表明该指令有效，故下一周期会往M寄存器插入气泡，同时465ns~475ns处，程序运行状态被置为HLT，故下一周期F寄存器被暂停，这与流水线控制逻辑对异常的预期处理一致。可以看到最后所有寄存器均处于稳定状态，且没有任何程序员可见状态在发生异常后被修改（注：RAS不是程序员可见状态，其不会被修改不属于预期），这验证了处理器异常处理逻辑的正确性。

#### 6.2.6 硬件结果展示



硬件测试结果

## 实验总结与心得

7.1实验总结

本次计算机组成原理课程设计历时四天（6月30日-7月3日），团队四名成员通过紧密协作，成功完成了基于RAS预测优化的Y86-64五级流水线处理器的设计与FPGA实现。项目不仅实现了预期的基础功能，更在性能优化方面取得了显著突破。

从功能实现角度来看，我们成功构建了完整的Y86指令集流水处理器，支持所有标准Y86指令的执行。五级流水线架构的实现确保了指令的并行执行能力，通过精心设计的数据前推机制和冒险处理逻辑，有效解决了流水线执行中的各种冲突问题。特别值得一提的是，我们创新性地引入了返回地址栈（RAS）预测机制，这一设计显著提升了函数调用密集型程序的执行效率。

从性能优化角度来看，RAS预测机制的引入带来了实质性的性能提升。通过将`ret`指令的执行延迟从传统的3个时钟周期降低到接近0周期，我们在硬件实现中采用了双向移位寄存器架构替代传统的寄存器堆+栈指针方案，最终测试，从475ns优化到415ns，延迟降低约12.6%，这一优化对提高系统工作频率具有重要意义。

从工程实践角度来看，项目成功地在Minisys FPGA平台上实现了完整的硬件验证。我们不仅完成了基本的处理器功能验证，还实现了丰富的调试和演示功能，包括LED状态指示、数码管周期计数显示、单拍/连续执行模式切换等。这些功能的实现不仅提升了系统的可用性，也为后续的调试和优化工作提供了有力支持。

#### 7.2 团队学习与成长

通过本次课程设计，我们不仅完成了预定的技术目标，更重要的是在团队协作、技术能力、工程实践等方面都获得了显著的成长。

**技术能力的全面提升**是最直观的收获。每个团队成员都在自己负责的任务中获得了技术积累，同时通过跨领域的协作也拓展了知识面。架构设计组成员深入掌握了处理器设计的核心技术，包括流水线架构、冒险处理、性能优化等；硬件验证组成员熟练掌握了FPGA开发流程、硬件调试技巧、系统集成方法等。

**工程实践能力的培养**是本次项目的重要成果。从需求分析到方案设计，从代码实现到硬件验证，从性能测试到系统优化，团队成员经历了完整的工程开发流程。这种实践经验对于培养工程思维、提高解决实际问题的能力具有重要意义。

**团队协作技能的发展**也是重要的收获。通过本次项目，我们学会了如何根据每个人的优势明确分工、如何在分工协作的框架下有效沟通、如何处理跨领域的技术问题、如何在时间压力下保持高质量的工作输出。这些软技能对于我们未来的职业发展同样重要。

## 8. 未来优化方向

### 8.1 单拍/连续模式切换功能的异步化

在目前的设计中，我们主要的设计思路是如果单击设定的按钮，就进行连续和单拍模式的切换。

具体代码逻辑是写在always(posedge clk or posedge rst)模块中，这样就导致了，模式切换必须在是综合功能上升沿和按钮单击同时进行时才能触发模式切换，但我们想要的是模式切换应该与时钟想脱离。优化的思路为：设计一个选择器，具体通过状态转移来实现，当到达1状态时切换为单拍模式，当到达3状态时切换为连续模式。然后2个状态之间存在1个其他的状态，如果处于1状态，来到一个时钟上升沿就切换到2状态，此时当且仅当单击按钮有效时才会转移到模式状态，这样即可将单击按钮与时钟独立开来。

### 8.2 性能优化方向

我们的设计使用总是选择分支的预测策略。研究表明这个策略的成功率大约为60%。稍微复杂一点的是反向选择、正向不选择的策略，当分支地址比下一条地址低时就预测选择分支，而分支地址比较高时，就预测不选择分支。这种策略的成功率大约为65%。这种改进源自一个事实，即循环是由后向分支结束的，而循环通常会执行多次。前向分支用于条件操作，而这种选择的可能性较小。对于预测是否选择分支的有效策略已经进行了广泛的研究。有的系统花费了大量硬件来解决这个任务。我们的设计只使用了简单的策略。后续可以考虑引入动态预测策略，从而提高分支预测的成功率。

### 8.3 功能扩展方向

我们的设计基于Y86-64指令集。事实上，Y86-64指令集为教学用指令集，功能较为局限，例如不支持乘除法，不支持浮点数，算数指令只有寄存器寻址，且仅支持64位数据单元，对操作系统也没有良好的接口。未来可以考虑扩展指令集，并基于Y86-64指令集开发简单的操作系统等，为用户提供更友好的交互接口。

## 9. 参考文献

[1] Bryant, R. E., & O'Hallaron, D. R. (2015). Computer Systems: A Programmer's Perspective (3rd ed.). Pearson.

## 10. 附录

### 10.1 成员心得总结

计子毅：

作为硬件组的一员，我全程参与了Minisys实验板的FPGA实现与调试工作，这段经历让我对硬件系统开发有了全新的认知。在项目初期，我负责通过分析Vivado综合报告，我们最终将逻辑资源使用率控制在43%，同时保证了50MHz的目标频率。这一过程让我深刻理解了时序约束文件（XDC）的重要性，特别是如何平衡时钟分频模块的周期约束与I/O延迟。在硬件调试阶段，我主导设计了板级状态指示系统。通过充分利用实验板的24个LED灯（红、黄、绿各8个），我们实现了三级可视化调试方案：红色LED组显示流水线异常状态，绿色LED组标记各阶段指令有效性，黄色LED组实时反馈RAS预测命中情况。这种硬件级的调试方法，相比仿真波形更直观高效。特别值得一提的是数码管显示功能的实现。我设计了扫描刷新机 ，将100MHz主时钟分频为2ms刷新周期，通过动态驱动两个4位七段数码管，稳定显示处理器周期计数。这个过程中，我们优化了段选信号(CA-CG/DP)和位选信号(A7-A0)的驱动时序，成功解决了早期版本存在的显示闪烁问题。在模式切换功能开发中，我创新性地改进了按键检测电路。通过将mode\_btn按键信号异步化处理，使其独立于系统时钟，解决了原设计需要时钟边沿同步触发的操作延迟问题。这个改进使得单步/连续模式的切换响应时间从3个周期缩短到1个周期。与软件组的协作也让我获益良多。这种跨组协作模式，让我理解了硬件设计必须充分考虑软件层的可测试性。这次课设经历重塑了我的工程思维：从最初只关注功能实现，到后期综合考虑时序收敛、功耗优化、调试便利性等多维指标。当最终看到处理器在实验板上稳定运行，数码管数字规律跳动时，那种将抽象设计转化为物理实物的成就感，是纯理论学习永远无法给予的。这段经历不仅巩固了我的数字电路基础，更培养了我作为硬件工程师必备的系统视角和严谨作风。

李思琪：

作为架构研究与方案设计组的成员，我在本次课程设计中主要负责返回地址预测和检测恢复机制的设计与实现，以及相关的冒险处理逻辑优化。这一经历让我对计算机组成原理有了更加深入和全面的理解。

对流水线技术的深度认知是我重要的收获之一。在理论学习中，通过精读CSAPP第四章，我了解了流水线的基本概念和工作原理，但真正的理解来自于实际的设计和代码实现过程。通过亲手构建五级流水线处理器，我深刻体会到了流水线设计的复杂性和精妙之处。

流水线不仅仅是将指令执行过程分解为多个阶段，更重要的是要处理各种复杂的相互依赖关系。数据冒险、控制冒险的处理需要精确的逻辑设计和细致的时序控制。

对性能优化的系统性思考是另一个重要收获。RAS预测机制的设计过程让我学会了如何系统性地分析性能瓶颈、如何设计有效的优化方案、如何在性能和复杂度之间取得平衡。

在分析`ret`指令的性能问题时，我不仅要考虑指令本身的执行特点，还要分析其在整个程序执行过程中的影响。通过量化分析，我发现`ret`指令虽然在指令总数中占比不高，但其执行延迟对整体性能的影响却很显著。这种系统性的分析方法让我学会了如何抓住关键问题，集中精力解决最重要的性能瓶颈。

对创新设计的实践体验更是珍贵的经历。从问题发现到方案设计，从理论分析到实际实现，从功能验证到性能优化，整个创新过程让我体验到了科研工作的完整流程。将学习到的“资源复用”和“流水线前推”思想用于具体实践设计，让我感受到创新的趣味性和成就感，也体会到创新不是凭空产生的，而是建立在对问题本质的深入理解基础上的。只有真正理解了问题的根源和约束条件，才能设计出真正有效的创新解决方案。如果没有对流水线架构的深刻理解和熟悉，也就很难洞察出创新点。

鹿一凡：作为硬件开发团队的核心成员，我全程参与了Y86-64五级流水线处理器的FPGA实现工作，这段经历让我对硬件系统开发有了全新的认知。在项目初期，我主要负责Minisys实验板的资源评估与分配，通过对XC7A100T FPGA的详细分析，我们最终将逻辑资源使用率控制在45%以内，同时保证了50MHz的目标工作频率。这一过程让我深刻理解了时序约束的重要性，特别是如何平衡时钟网络的布局与关键路径的优化。

在具体实现环节，我主导开发了系统的时钟管理模块。我们创新性地采用了混合时钟方案：将100MHz的主时钟通过PLL生成50MHz的核心时钟用于处理器流水线，同时分频出1KHz的低频时钟用于外设控制。这种设计既确保了处理器的运算性能，又实现了稳定的外设交互。

外设接口的实现是我的另一个重点工作。我设计了完整的数码管显示系统，采用动态扫描技术驱动两个4位七段数码管。通过精确控制位选信号（A7-A0）和段选信号（CA-CG/DP）的时序关系，我们实现了稳定的周期计数显示功能。特别值得一提的是，我们优化了刷新算法，将显示刷新率提高到60Hz，完全消除了人眼可察觉的闪烁现象。

LED状态指示系统的开发也颇具挑战性。我们充分利用实验板的24个LED灯（红、黄、绿各8个），设计了三重状态指示方案：红色LED组显示流水线异常和冒险情况，绿色LED组标记各流水线阶段的活动状态，黄色LED组则专门用于RAS预测的反馈。这种可视化的调试方案大大提升了问题定位的效率。

这次项目经历让我对硬件开发的认知有了质的飞跃：从最初只关注功能实现，到后期综合考虑时序收敛、功耗优化、电磁兼容等多维指标。当最终看到处理器在实验板上稳定运行SPEC测试程序，数码管清晰地显示着1.12的CPI时，那种将抽象设计转化为物理实物的成就感无以言表。这段经历不仅夯实了我的数字电路基础，更培养了我作为硬件工程师必备的系统思维和工程素养，为未来参与更复杂的SoC设计奠定了坚实基础。

陈子容：通过本次计组课程设计中对返回地址栈（RAS）的完整实现，我对处理器分支预测机制的设计挑战有了更本质的理解。初期低估了投机执行对硬件状态机的影响，在直接移植基础RAS方案时遭遇了预测错误污染问题——错误路径上的call/ret指令导致栈指针异常偏移，引发函数返回地址丢失。这个教训让我意识到硬件设计必须遵循状态操作的原子性原则，最终通过延迟操作至M阶段提交，并引入全局冲刷信号才彻底解决。

实践过程中最深刻的认知是硬件时序与软件优化的关联性。在计算机网络实验中，我对CRC库函数的实现方式感到疑惑，函数将循环展开，增加了代码行数，降低了代码的可读性，这令我十分不解。直到我亲手设计并实现过一个完整的流水处理器，亲眼看到程序在系统上运行，我才意识到，使用循环展开后，分支指令数量下降35%，相应的预测错误率降低22%。这证实了计算机系统中层间协作的倍增效应：一条高级语言的结构优化，可能消除底层数十个时钟周期的冒险开销。未来开发高性能程序时，将优先考虑用数据级并行替代条件分支。

这次实践积累的波形分析经验将成为后续学习超标量架构的基础，也将助力我后续更好地为北邮RROS项目出一份力。真正的硬件思维不仅限于高速运算，而是在高速运算中同样高度安全。

### 10.2 工作日志

6月30日

工具链部署与硬件验证组（计子毅+鹿一凡）

• 完成Vivado/ISE开发环境配置

• 确认硬件调试链路及工具链功能正常

架构研究与方案设计组（李思琪+陈子容）

• 分析指令集编码、五级流水线结构及数据通路 确定数据冒险、及控制冒险 的初步解决方案

• 提供Y86 Verilog代码（PIPE）——陈子容

• 提出开发汇编器 计划通过用户数据接口实现机器码直写芯片的执行模式——陈子容

7月1日

工具链部署与硬件验证组（计子毅+鹿一凡）

• 调整Y86架构代码，完成在Vivado开发平台上的适配。 将适配后的代码烧录至Minisys开发板。

架构研究与方案设计组（李思琪+陈子容）

• 仿真时 通过观察波形图 发现寄存器文件读出的valA 是不定状态（X），xor等运算无法进行，则导致后续程序错误，解决方法：增加复位信号，将寄存器预设为初始值——陈子容

• 基于新增的冒险处理逻辑，修正前期非流水线版本代码中的相关错误(assign {f rA,f rB}=f ins[15:8]更改为assign {f rA,f rB} =need regids ?f ins[15:8]:{RNONE,RNONE} ;) （∵加载/使用冒险的检测依赖于源操作数寄存器标识符与加载指令目标寄存器的比对，所以要避免rarb存虚假的数据 ）——李思琪

7月2日

工具链部署与硬件验证组（计子毅+鹿一凡）

• 调整Y86架构代码，完成在Vivado开发平台上的适配。将适配后的代码烧录至Minisys开发板。发现并纠正了课件上硬件对应端口错误，实现了系统时钟的分频以将Y86运行结果完好展示在小灯上。 测试RAS 代码，475ns → 415ns

架构研究与方案设计组（李思琪+陈子容）

在PIPE的基础上，新增 返回地址预测逻辑 ，增加返回地址栈RAS(return address stack）Verilog代码实现；新增 返回地址预测错误判断 和 恢复逻辑，对应修改 流水线控制逻辑——李思琪

7月3日

工具链部署与硬件验证组（计子毅+鹿一凡）

• 增加了分频后系统时钟计数功能，将系统时钟分频为LED刷新频率，使计数结果在LED屏中显示，并在停机后停止计数，rst后置零。实现了单拍执行模式，增加(单拍/连续)模式切换功能

架构研究与方案设计组（李思琪+陈子容）

• 优化栈的底层实现 寄存器堆+栈指针 → 双向移位寄存器 ——陈子容

### 10.3贡献度表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **已实现的功能（在实现的功能前打勾、扩展实验自己写功能），提交作业时一并提交** | | | | | | |
| **拓展题目** | **基本功能** | ◊ 流水线处理器  ◊ 支持标准五级流水线  ◊ 支持Y86指令集  ◊ 支持暂停，转发，气泡、异常处理 | | **附加功能** | ◊ 增设返回地址栈（RAS）  ◊ 增设L1缓存  ◊ 实现单拍/连续执行模式切换  ◊ 提供显示逻辑 | |
| **学 号** | **姓名** | | **承担的工作** | | | **贡献度**  **（总共100%）** |
| **2023210710** | **陈子容** | | **提供并调试PIPE代码**  **编写并调试RAS代码**  **实现L1缓存** | | | **25%** |
| **2023212122** | **李思琪** | | **设计并优化RAS及相关控制逻辑**  **记录工作日志**  **撰写技术方案报告** | | | **25%** |
| **2023212304** | **鹿一凡** | | **调试硬件与开发平台**  **实现显示逻辑**  **撰写硬件部分文档** | | | **25%** |
| **2023212872** | **计子毅** | | **调试硬件与开发平台**  **实现单拍/连续执行模式切换**  **撰写硬件部分文档** | | | **25%** |