同济大学计算机科学与技术学院

计算机系统结构课程实验总结报告

实验名称	简单的流水线 CPU 设计与性能分析
学　　号	2351579
姓　　名	程浩然
专　　业	计算机科学与技术
授课教师	秦国锋
日　　期	2025 年 11 月 23 日

目录

第一部分 实验环境部署与硬件配置说明

本实验基于 MIPS 架构的 5 级流水线 CPU 设计。硬件平台采用 Xilinx Vivado 设计套件实现，主要包括：

- 流水线处理器核心：包括 IF、ID、EX、MEM、WB 五个阶段
- 流水线寄存器：IF/ID、ID/EX、EX/MEM、MEM/WB 寄存器
- 存储器：指令存储器 (IMEM) 和数据存储器 (DMEM)
- 功能部件：算术逻辑单元 (ALU)、寄存器文件、分支预测单元等

实验环境使用 ModelSim 进行仿真验证，通过 run_cpu_tests.do 脚本自动化运行测试用例。

第二部分 实验的总体结构

一. 5 级指令流水线的总体结构

本实验实现了 MIPS 架构的 5 级流水线处理器，包括以下 5 个阶段：

1. **IF (Instruction Fetch) - 指令获取阶段**：从指令存储器获取指令，更新程序计数器 (PC)。
2. **ID (Instruction Decode) - 指令译码阶段**：译码指令，从寄存器文件读取操作数，检测分支指令。
3. **EX (Execute) - 执行阶段**：在 ALU 中执行操作，计算内存访问地址。
4. **MEM (Memory Access) - 内存访问阶段**：访问数据存储器，执行内存读/写操作。
5. **WB (Write Back) - 写回阶段**：将结果写回寄存器文件。

第三部分 总体架构部件的解释说明

一. 5 级指令流水线总体结构部件的解释说明

3.1.1 IF/ID 寄存器

IF/ID 寄存器连接指令获取阶段和指令译码阶段，保存从 IF 阶段传递到 ID 阶段的信息，包括：

- 指令本身
- 当前 PC 值
- 目标 PC 值

3.1.2 ID/EX 寄存器

ID/EX 寄存器连接指令译码阶段和执行阶段，包含：

- 译码后的指令
- 操作数 A 和 B
- ALU 控制信号

3.1.3 EX/MEM 寄存器

EX/MEM 寄存器连接执行阶段和内存访问阶段，包含：

- ALU 计算结果
- 内存访问控制信号
- 内存写入数据

3.1.4 MEM/WB 寄存器

MEM/WB 寄存器连接内存访问阶段和写回阶段，包含：

- 内存访问结果
- 写入寄存器的目标地址
- 写使能信号

第四部分　实验仿真过程

一. 5 级指令流水线的仿真过程

仿真过程使用 ModelSim 工具进行，主要包括以下步骤：

1. 编译所有 Verilog 源文件
2. 加载测试平台和待测 CPU 模块

3. 为不同测试用例加载相应的指令序列

4. 运行仿真并记录每个时钟周期的 PC、指令和寄存器状态

5. 将仿真结果与软件参考模型进行比较验证

自动化测试脚本 run_cpu_tests.do 会依次运行多个测试用例，包括 ADDI、ADDIU、LW/SW、BEQ、BNE、SLL、SUBU、SLTU 等指令的测试。

第五部分　实验仿真的波形图及某时刻寄存器值的物理意义

一.　5 级指令流水线的波形图及某时刻寄存器值的物理意义

在仿真过程中，每个时钟周期都会记录 CPU 的内部状态，包括：

- PC 值：当前指令的程序计数器值
- 指令：当前执行的指令
- 寄存器文件：32 个通用寄存器的值

寄存器值的物理意义：

- $0：始终为 0，是 MIPS 架构的固定约定
- $1-$31：通用寄存器，用于存储操作数、结果和地址

通过分析波形图，可以观察到流水线的并行执行特性，在稳定状态时，每个时钟周期都有一个指令在不同阶段执行。

第六部分　流水线 CPU 实验性能验证模型

一.　实验性能验证模型：比萨塔摔鸡蛋游戏

设计 C 语言算法验证该模型：

```c
// 比萨塔摔鸡蛋游戏验证模型的C算法
#include <stdio.h>

// 算法：使用二分搜索策略寻找鸡蛋的耐摔值
int egg_drop_simulation(int floors, int durability) {
    int drops = 0;          // 摔的总次数
    int eggs_used = 0;      // 使用的总鸡蛋数
```

```
    int broken_eggs = 0;   // 摔破的鸡蛋总数
    int current_floor = 1;
    int last_safe_floor = 0;
    int egg_broken = 0;    // 最后一个鸡蛋是否摔破

    // 首先使用二分搜索缩小范围
    int low = 1;
    int high = floors;
    int mid;

    while (low <= high) {
        drops++;
        eggs_used++;
        mid = (low + high) / 2;

        // 检查在mid层摔鸡蛋的情况
        if (mid <= durability) {
            // 鸡蛋没破，安全楼层提高
            last_safe_floor = mid;
            low = mid + 1;
        } else {
            // 鸡蛋破了，需要降低搜索范围
            broken_eggs++;
            high = mid - 1;
        }

        // 如果鸡蛋破了，我们需要线性搜索
        if (broken_eggs > 0) {
            // 从上次安全楼层+1开始逐层测试
            int test_floor = last_safe_floor + 1;
            while (test_floor < mid) {
                drops++;
                eggs_used++;
                if (test_floor > durability) {
                    // 鸡蛋破了
                    broken_eggs++;
                } else {
                    // 鸡蛋没破
                    last_safe_floor = test_floor;
                }
```

```
48                  test_floor++;
49              }
50              break;
51          }
52      }
53
54      // 最后一个鸡蛋的状态
55      if (broken_eggs > 0 && last_safe_floor + 1 > durability) {
56          egg_broken = 1;
57      } else {
58          egg_broken = 0;
59      }
60
61      // 输出结果
62      printf("摔的总次数: %d\n", drops);
63      printf("使用的总鸡蛋数: %d\n", eggs_used);
64      printf("摔破的鸡蛋总数: %d\n", broken_eggs);
65      printf("最后摔的鸡蛋是否摔破: %s\n", egg_broken ? "是" : "否");
66
67      return broken_eggs;  // 返回摔破的鸡蛋数
68  }
69
70  // 计算总成本f = m*p1 + n*p2 + h*p3
71  void calculate_cost(int floors_up, int floors_down, int broken_eggs,
        int p1, int p2, int p3) {
72      int total_cost = floors_up * p1 + floors_down * p2 + broken_eggs
            * p3;
73      printf("总成本 f = m*p1 + n*p2 + h*p3 = %d*%d + %d*%d + %d*%d = %
            d\n",
74              floors_up, p1, floors_down, p2, broken_eggs, p3,
                    total_cost);
75  }
76
77  int main() {
78      int durability = 25;  // 鸡蛋耐摔值（示例）
79      int floors = 100;     // 比萨塔层数（示例）
80
81      printf("比萨塔摔鸡蛋游戏验证模型\n");
82      printf("鸡蛋耐摔值: %d, 比萨塔层数: %d\n\n", durability, floors);
83
```

```
    // 物质匮乏时期
    printf("物质匮乏时期 (p1=2, p2=1, p3=4):\n");
    int broken_eggs_1 = egg_drop_simulation(floors, durability);
    calculate_cost(50, 20, broken_eggs_1, 2, 1, 4);  // 示例值
    printf("\n");

    // 人力成本增长时期
    printf("人力成本增长时期 (p1=4, p2=1, p3=2):\n");
    int broken_eggs_2 = egg_drop_simulation(floors, durability);
    calculate_cost(30, 15, broken_eggs_2, 4, 1, 2);  // 示例值
    printf("\n");

    return 0;
}
```

二. MIPS 汇编程序

将上述 C 算法转换为 MIPS 汇编程序：

```
# 比萨塔摔鸡蛋游戏验证模型 - MIPS汇编实现
#
# 使用寄存器约定:
# $a0 - 塔的总层数 (floors)
# $a1 - 鸡蛋耐摔值 (durability)
#
# $t0 - drops (摔的总次数)
# $t1 - eggs_used (使用的总鸡蛋数)
# $t2 - broken_eggs (摔破的鸡蛋总数)
# $t3 - current_floor (当前测试的楼层)
# $t4 - last_safe_floor (最后一个安全的楼层)
# $t5 - low (二分搜索的下界)
# $t6 - high (二分搜索的上界)
# $t7 - mid (二分搜索的中间值)

.text
.globl main

main:
    # 初始化值（示例）
    li $a0, 100         # 塔的总层数
```

```
    li $a1, 25          # 鸡蛋耐摔值

    # 跳转到egg_drop_simulation函数
    jal egg_drop_simulation

    # 准备计算成本
    li $a2, 50          # 楼上楼层总数 (示例值)
    li $a3, 20          # 楼下楼层总数 (示例值)

    # 物质匮乏时期 (p1=2, p2=1, p3=4)
    li $t8, 2           # p1
    li $t9, 1           # p2
    li $t10, 4          # p3
    jal calculate_cost
    move $s0, $v0       # 保存成本结果

    # 人力成本增长时期 (p1=4, p2=1, p3=2)
    li $t8, 4           # p1
    li $t9, 1           # p2
    li $t10, 2          # p3
    jal calculate_cost
    move $s1, $v0       # 保存成本结果

    # 程序结束
    li $v0, 10
    syscall

# 鸡蛋摔破模拟函数
# 输入: $a0 = 塔的总层数, $a1 = 鸡蛋耐摔值
# 输出: $v0 = 摔破的鸡蛋数
egg_drop_simulation:
    addi $sp, $sp, -20      # 分配栈空间
    sw $ra, 16($sp)         # 保存返回地址
    sw $s0, 12($sp)         # 保存其他寄存器
    sw $s1, 8($sp)
    sw $s2, 4($sp)
    sw $s3, 0($sp)

    # 初始化变量
    li $t0, 0               # drops = 0
```

```
    li $t1, 0               # eggs_used = 0
    li $t2, 0               # broken_eggs = 0
    li $t3, 1               # current_floor = 1
    li $t4, 0               # last_safe_floor = 0
    li $t5, 1               # low = 1
    move $t6, $a0           # high = floors

binary_search_loop:
    # 检查循环条件: low <= high
    bgt $t5, $t6, linear_search_loop

    # 计算 mid = (low + high) / 2
    add $t7, $t5, $t6
    srl $t7, $t7, 1         # 右移1位等于除2

    # 增加计数器
    addi $t0, $t0, 1        # drops++
    addi $t1, $t1, 1        # eggs_used++

    # 检查 mid <= durability (鸡蛋是否摔破)
    bgt $t7, $a1, egg_breaks

    # 鸡蛋没破: last_safe_floor = mid; low = mid + 1
    move $t4, $t7
    addi $t5, $t7, 1
    j binary_search_loop

egg_breaks:
    # 鸡蛋破了: broken_eggs++; high = mid - 1
    addi $t2, $t2, 1
    addi $t6, $t7, -1

    # 检查是否已经有过鸡蛋摔破
    bne $t2, $zero, linear_search_loop
    j binary_search_loop

linear_search_loop:
    # 线性搜索: 从 last_safe_floor+1 到 mid-1
    addi $t8, $t4, 1        # test_floor = last_safe_floor + 1
    move $t9, $t7           # test_floor <= mid-1
```

```
102
103 linear_search_iter:
104     bge $t8, $t9, finished_search
105
106     # 增加计数器
107     addi $t0, $t0, 1        # drops++
108     addi $t1, $t1, 1        # eggs_used++
109
110     # 检查 test_floor > durability
111     bgt $t8, $a1, linear_egg_breaks
112
113     # 鸡蛋没破: last_safe_floor = test_floor
114     move $t4, $t8
115     addi $t8, $t8, 1        # test_floor++
116     j linear_search_iter
117
118 linear_egg_breaks:
119     # 鸡蛋破了: broken_eggs++
120     addi $t2, $t2, 1
121     addi $t8, $t8, 1        # test_floor++
122     j linear_search_iter
123
124 finished_search:
125     # 检查最后摔的鸡蛋是否摔破
126     addi $t3, $t4, 1
127     ble $t3, $a1, end_simulation
128
129     # 最后一个鸡蛋摔破了
130     # 这里可以设置标志但当前不需要
131
132 end_simulation:
133     # 将结果放入返回寄存器
134     move $v0, $t2           # 返回 broken_eggs
135
136     # 恢复寄存器
137     lw $s3, 0($sp)
138     lw $s2, 4($sp)
139     lw $s1, 8($sp)
140     lw $s0, 12($sp)
141     lw $ra, 16($sp)
```

```
142     addi $sp, $sp, 20       # 释放栈空间
143
144     jr $ra                  # 返回
145
146 # 计算总成本函数
147 # 输入: $a2 = 上楼层数 (floors_up)
148 #       $a3 = 下楼层数 (floors_down)
149 #       $v0 = 摔破鸡蛋数 (broken_eggs)
150 #       $t8 = p1, $t9 = p2, $t10 = p3
151 # 输出: $v0 = 总成本 f
152 calculate_cost:
153     addi $sp, $sp, -16      # 分配栈空间
154     sw $ra, 12($sp)         # 保存返回地址
155
156     # 计算 f = floors_up * p1 + floors_down * p2 + broken_eggs * p3
157     mult $a2, $t8           # floors_up * p1
158     mflo $t0                # $t0 = floors_up * p1
159
160     mult $a3, $t9           # floors_down * p2
161     mflo $t1                # $t1 = floors_down * p2
162
163     mult $v0, $t10          # broken_eggs * p3
164     mflo $t2                # $t2 = broken_eggs * p3
165
166     add $t0, $t0, $t1       # $t0 = floors_up * p1 + floors_down * p2
167     add $v0, $t0, $t2       # $v0 = 总成本 f
168
169     # 恢复寄存器
170     lw $ra, 12($sp)
171     addi $sp, $sp, 16       # 释放栈空间
172
173     jr $ra                  # 返回
```

第七部分　实验验算程序下板测试过程与实现

下板测试主要步骤包括：

1. 将设计下载到 FPGA 开发板

2. 配置测试向量和时钟源

3. 运行测试程序并监测输出结果

4. 验证功能正确性

第八部分 流水线的性能指标定性分析

一. 静态流水线的性能指标定性分析

8.1.1 吞吐率 (Throughput)

理想情况下，5 级流水线的吞吐率为 1 条指令/时钟周期，即在稳定状态下每个时钟周期可以完成一条指令。但在实际应用中，由于数据相关、控制相关和结构冲突的存在，吞吐率会有所降低。

8.1.2 加速比 (Speedup)

理论上，5 级流水线的理想加速比为 5，但由于数据相关、控制相关和结构冲突，实际加速比会小于理论值。

8.1.3 效率 (Efficiency)

流水线效率 = 实际加速比 / 理论加速比，理想值为 1，实际值小于 1。流水线效率受相关冲突、分支预测准确性等因素影响。

8.1.4 相关与冲突分析

- **数据相关**：通过数据前递技术解决，当无法前递时需要插入气泡（停顿周期）
- **控制相关**：通过分支预测和延迟槽技术处理
- **结构冲突**：通过增加硬件资源或插入停顿周期解决

8.1.5 CPU 的运行时间及存储器空间的使用

流水线 CPU 在时间上通过并行处理多条指令提高了指令执行效率，在空间上需要额外的流水线寄存器来存储中间结果，但总体资源开销是可接受的。

第九部分 总结与体会

通过对 5 级流水线 CPU 设计的学习和实践，我深刻理解了流水线技术在提高 CPU 性能方面的重要作用。通过将指令执行过程分解为多个阶段，并行处理多条指令，可以

显著提高处理器的吞吐率。在设计过程中，需要处理各种相关和冲突问题，如数据相关、控制相关和结构冲突，这需要采用前递、分支预测、流水线停顿等技术来解决。

在本次实验中，我们实现了包括 ADD、ADDU、ADDI、ADDIU、SLL、LW、SW、SUBU、BNE、BEQ、SLTU 和 HALT 在内的 12 条指令的流水线处理器。通过数据前递技术解决了数据相关问题，通过分支预测机制处理控制相关问题。

流水线技术虽然提高了指令执行效率，但也引入了复杂性，如数据冒险、控制冒险和结构冒险的处理。通过本次实验，我不仅掌握了流水线的基本概念和设计方法，还学会了如何分析和解决流水线中的各种冲突问题，对计算机系统结构有了更深入的理解。

第十部分　附件（所有程序）

一. 5 级指令流水线的设计程序

10.1.1 顶层模块 (sccomp_dataflow.v)

10.1.2 CPU 核心模块 (cpu.v)

10.1.3 算术逻辑单元 (alu.v)

10.1.4 流水线寄存器模块 (IF_ID.v, ID_EX.v, EX_MEM.v, MEM_WB.v)

10.1.5 寄存器文件 (regfile.v)

10.1.6 定义文件 (def.v)

10.1.7 测试平台 (_246tb_ex10_tb.v)

10.1.8 自动化测试脚本 (run_cpu_tests.do)