# 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

本实验围绕 MIPS 架构的 5 级流水线 CPU 展开设计，硬件平台基于 Xilinx Vivado 设计套件实现，核心硬件组成包括：

- 流水线处理器核心：划分为取指 (IF)、译码 (ID)、执行 (EX)、访存 (MEM)、写回 (WB) 五个阶段
- 流水线寄存器：包含 IF/ID、ID/EX、EX/MEM、MEM/WB 四级流水寄存器
- 存储模块：指令存储器 (IMEM) 与数据存储器 (DMEM)
- 功能部件：算术逻辑单元 (ALU)、寄存器文件、分支预测单元等

实验采用 ModelSim 进行仿真验证，通过自动化测试脚本 run_cpu_tests.do 批量运行测试用例，完成功能与性能的验证。

## 一. 指令集实现

本实验复用计算机组成原理课程中的指令集设计方案，指令与 MIPS 架构的映射关系如下表所示：

| 指令 | MIPS 实现方式 |
|---|---|
| ADD | 采用 add、addi、addiu、addu 指令实现 |
| NOP | 转义为 sll \$0,\$0,0 指令 |
| HALT | 自定义新增指令实现 |
| LOAD | 复用 lw 指令实现 |
| STORE | 复用 sw 指令实现 |
| CMP | 通过 sltu \$rd,\$rs,\$rt 或 subu \$0,\$rs,\$rt 指令实现 |
| BZ | 转义为 beq \$0,\$r,\$label 指令 |
| BN | 转义为 bne \$0,\$r,\$label 指令 |

表 1: 指令转义关系表

设计中手绘 cpu 架构图如下

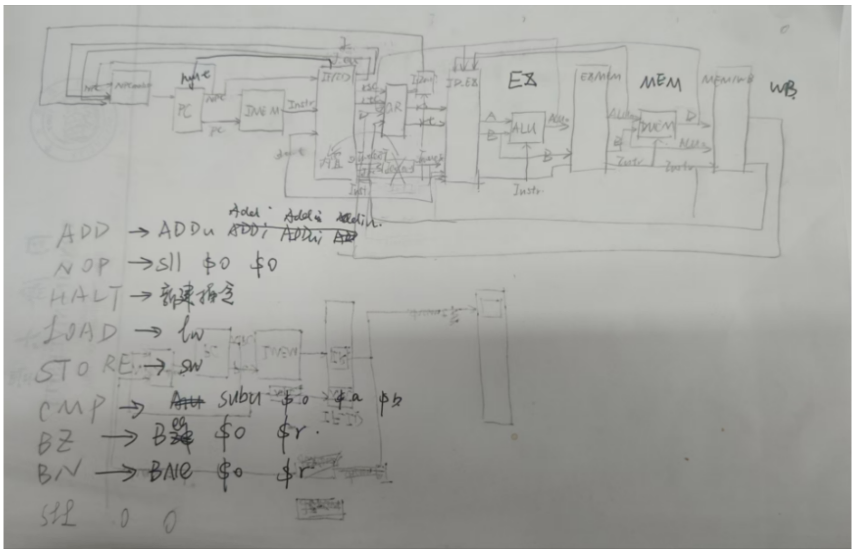

PC
IMEM
Instr
IF/ID
ID/EX
EX
ALU
EX/MEM
MEM
DMEM
MEM/WB
WB
ADD → ADDu ~~ADDi~~ ADDu
NOP → sll $0 $0
HALT → 新建指令
LOAD → lw
STORE → sw
CMP → subu $0 $a $b
BZ → BEQ $0 $r.
BN → BNE $0 $r
sll 0 0

图 1:

## 二. 分支指令设计

分支指令的执行周期以指令读取为第 0 周期，具体处理逻辑为：

- 第 0 周期：IF_ID 寄存器直接输出 PC_bobl 信号为 1，触发分支检测
- 第 1 周期：程序计数器 PC 保持当前值不变，同时指令存储器 IMEM 输出 NOP 指令，插入流水线气泡

# 第二部分 实验的总体结构

## 一. 5 级指令流水线的总体结构

本实验实现的 MIPS 架构 5 级流水线处理器，将指令执行过程拆解为以下五个阶段，各阶段的核心功能如下：

1. **取指阶段 (IF, Instruction Fetch)**：从指令存储器中读取当前 PC 指向的指令，并完成程序计数器的自增更新。
2. **译码阶段 (ID, Instruction Decode)**：对取指阶段获取的指令进行译码，从寄存器文件中读取操作数，并检测分支指令的跳转条件。
3. **执行阶段 (EX, Execute)**：在算术逻辑单元 ALU 中执行译码后的运算操作，同时计算内存访问的有效地址。
4. **访存阶段 (MEM, Memory Access)**：根据执行阶段的结果，完成数据存储器的读/写操作，实现数据的访存交互。
5. **写回阶段 (WB, Write Back)**：将执行结果或访存数据写回寄存器文件，完成指令的最终执行流程。

# 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 寄存器作为取指与译码阶段的衔接部件，主要用于暂存从 IF 阶段传递至 ID 阶段的关键信息，具体包括：

- 取指阶段获取的指令编码
- 当前程序计数器 PC 的数值
- 分支指令的目标 PC 地址

#### 3.1.2 ID/EX 寄存器

ID/EX 寄存器承担译码与执行阶段的信息传递功能，暂存的核心数据包括：

- 译码后的指令操作码与操作数信息
- 从寄存器文件中读取的操作数 A 和操作数 B
- 控制单元生成的 ALU 运算控制信号

#### 3.1.3 EX/MEM 寄存器

EX/MEM 寄存器连接执行与访存阶段，主要存储执行阶段的运算结果与访存控制信息，包括：

- ALU 的运算结果与内存访问地址
- 数据存储器的读/写控制信号
- 待写入数据存储器的原始数据

#### 3.1.4 MEM/WB 寄存器

MEM/WB 寄存器是访存与写回阶段的桥梁，暂存的信息用于完成最终的写回操作，包括：

- 数据存储器的读取结果
- 待写入寄存器文件的目标寄存器地址
- 寄存器写回的使能控制信号

# 第四部分　实验仿真过程

## 一. 5 级指令流水线的仿真过程

本实验采用 ModelSim 工具完成流水线 CPU 的仿真验证，具体仿真流程分为以下步骤：

1. 编译所有 Verilog 硬件描述语言源文件，生成可仿真的模块库
2. 加载测试平台模块与待测 CPU 核心模块，建立仿真拓扑
3. 为不同测试场景加载对应的指令序列，配置仿真激励
4. 运行仿真流程，实时记录每个时钟周期的 PC 值、指令内容与寄存器状态

5. 将仿真输出结果与软件参考模型的计算结果对比，验证功能正确性

自动化测试脚本 run_cpu_tests.do 会按序执行多组测试用例，覆盖 ADDI、ADDIU、LW/SW、BEQ、BNE、SLL、SUBU、SLTU 等核心指令的功能验证。

## 二. 启动测试与执行过程

根据实验配套的 readme.md 文档说明，通过以下指令启动仿真测试：

```
vsim -c -do "do run_cpu_tests.do; quit" >log
```

该命令以批处理模式运行 ModelSim，自动执行 run_cpu_tests.do 脚本并将仿真日志输出至 log 文件，便于后续结果分析。

## 三. 分支预测与冲突处理机制

### 4.3.1 分支指令处理

分支指令的流水线处理机制以指令读取为第 0 周期，具体逻辑为：

- 第 0 周期：IF_ID 寄存器直接输出 PC_bobl 信号为 1，触发分支检测逻辑
- 第 1 周期：程序计数器 PC 保持当前值，指令存储器 IMEM 输出 NOP 指令，插入流水线气泡

### 4.3.2 冲突检测与处理

流水线冲突的检测与处理流程如下：

1. 当检测到流水线冲突时，PC 暂停一个时钟周期更新，向 ID_EX 模块传递 NOP 指令，完成一次流水线冒泡
2. 冲突检测在译码 (ID) 阶段完成，检测到冲突后激活 detect_conflct 信号
3. PC、指令存储器 IMEM 与 IF/ID 寄存器在冲突周期内保持原有 PC 值与指令内容，确保流水线稳定

## 四. 写锁设计机制

寄存器文件的写锁设计是解决流水线数据冲突的核心机制，具体实现逻辑如下：

1. 当指令流经寄存器文件时，为待写入的寄存器添加写锁，直至写回阶段完成后释放写锁
2. 写操作之间不存在冲突，因为后发的写指令必然在先行的写指令之后执行

3. 读操作与写操作可能产生冲突，此类冲突需在 ALU 或 DMEM 模块中通过数据重定向解决
4. 为每个写锁配置时长为 3 的定时器，定时器归零时自动释放写锁（因数据需 3 个周期完成传递）
5. 若冲突无法通过重定向解决，则暂停 PC 更新一个周期，向 ID_EX 模块传递 NOP 指令，完成一次流水线冒泡
6. 写锁寄存器 reglock 的低 2 位为计时器，高 2 位标识锁的类型（ALU 型占用或 DMEM 型占用）
7. 冲突检测在译码 (ID) 阶段完成，检测到冲突后激活 detact_confict 信号，PC、IMEM 与 IF/ID 寄存器保持原有值一个周期

# 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，ModelSim 会实时记录 CPU 内部的关键信号波形，核心监测内容包括：

- PC 值：当前正在执行指令的程序计数器地址，反映指令的执行流
- 指令编码：当前流水线各阶段的指令内容，体现指令的执行进度
- 寄存器文件：32 个通用寄存器的实时数值，反映数据的运算与传递过程

MIPS 架构中寄存器值的物理意义明确：

- $0 寄存器：硬件层面固定为 0，是 MIPS 架构的固有约定，用于零值传递与空操作
- $1-$31 寄存器：通用寄存器，用于存储运算操作数、中间结果与内存访问地址

通过分析仿真波形图，可直观观察到流水线的并行执行特性：在稳定运行状态下，每个时钟周期均有一条指令在不同流水线阶段执行，实现指令的流水化处理。

# 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

为验证流水线 CPU 的性能，设计如下 MIPS 语言算法实现比萨塔摔鸡蛋游戏的仿真模型：

将上述 C 语言算法转换为 MIPS 汇编程序，实现流水线 CPU 的性能验证，代码如下：

```
addiu $s0, $zero, 100    # building_height = 100
addiu $s1, $zero, 42     # egg_durability = 42

# 物质匮乏时期成本参数
addiu $s2, $zero, 2      # scarcity_p1 = 2
addiu $s3, $zero, 1      # scarcity_p2 = 1
addiu $s4, $zero, 4      # scarcity_p3 = 4

# 人力成本增长时期成本参数
addiu $s5, $zero, 4      # labor_p1 = 4
addiu $s6, $zero, 1      # labor_p2 = 1
addiu $s7, $zero, 2      # labor_p3 = 2

# 开始模拟物质匮乏时期
# 初始化游戏变量
addiu $t0, $zero, 0      # total_drops = 0
addiu $t1, $zero, 0      # total_eggs_broken = 0
addiu $t2, $zero, 0      # last_egg_broken = 0
addiu $t3, $zero, 0      # total_cost = 0
addiu $t4, $zero, 1      # low = 1
addiu $t5, $zero, 100    # high = building_height
addiu $t6, $zero, 0      # current_floor = 0
addiu $t7, $zero, 2      # eggs_remaining = 2
addiu $t8, $zero, 0      # up_floors = 0
addiu $t9, $zero, 0      # down_floors = 0

# 物质匮乏时期游戏循环
scarcity_loop:
# 检查循环条件: low <= high && eggs_remaining > 0
sltu $at, $t5, $t4       # high < low?
bne $at, $zero, scarcity_end
beq $t7, $zero, scarcity_end

# 计算mid = (low + high) / 2
addu $at, $t4, $t5
```

```
srl $at, $at, 1

# 计算上下楼
sltu $k0, $t6, $at       # current_floor < mid?
bne $k0, $zero, scarcity_go_up

# 下楼情况
subu $k0, $t6, $at
addu $t9, $t9, $k0
beq $zero, $zero, scarcity_update_pos

scarcity_go_up:
# 上楼情况
subu $k0, $at, $t6
addu $t8, $t8, $k0

scarcity_update_pos:
# current_floor = mid
addiu $t6, $at, 0

# total_drops++
addiu $t0, $t0, 1

# 检查鸡蛋是否摔破
sltu $k0, $s1, $at       # egg_durability < mid?
bne $k0, $zero, scarcity_egg_broken

# 鸡蛋没破
addiu $t2, $zero, 0      # last_egg_broken = 0
addiu $t4, $at, 1        # low = mid + 1
beq $zero, $zero, scarcity_continue

scarcity_egg_broken:
# 鸡蛋摔破
addiu $t1, $t1, 1        # total_eggs_broken++
addiu $t7, $t7, -1       # eggs_remaining--
addiu $t2, $zero, 1      # last_egg_broken = 1
addiu $t5, $at, -1       # high = mid - 1

scarcity_continue:
beq $zero, $zero, scarcity_loop

scarcity_end:
# 计算物质匮乏时期总成本: up_floors*p1 + down_floors*p2 + eggs_broken*p3
addiu $k0, $zero, 0      # temp_cost = 0

# 计算 up_floors * p1 (加法模拟乘法)
addiu $k1, $zero, 0      # 乘法计数器
scarcity_mult1_loop:
beq $t8, $zero, scarcity_mult1_done
addu $k0, $k0, $s2
addiu $t8, $t8, -1
beq $zero, $zero, scarcity_mult1_loop
scarcity_mult1_done:

# 计算 down_floors * p2
addiu $k1, $zero, 0
scarcity_mult2_loop:
beq $t9, $zero, scarcity_mult2_done
addu $k0, $k0, $s3
addiu $t9, $t9, -1
beq $zero, $zero, scarcity_mult2_loop
scarcity_mult2_done:

# 计算 eggs_broken * p3
addiu $k1, $zero, 0
scarcity_mult3_loop:
beq $t1, $zero, scarcity_mult3_done
addu $k0, $k0, $s4
addiu $t1, $t1, -1
beq $zero, $zero, scarcity_mult3_loop
scarcity_mult3_done:

# 保存物质匮乏时期结果
```

```
addiu $t8, $t0, 0         # total_drops -> $t8
addiu $t9, $t1, 0         # total_eggs_broken -> $t9
addiu $k0, $t2, 0         # last_egg_broken -> $k0
addiu $k1, $k0, 0         # total_cost -> $k1

# 开始模拟人力成本增长时期
# 重新初始化游戏变量
addiu $t0, $zero, 0       # total_drops = 0
addiu $t1, $zero, 0       # total_eggs_broken = 0
addiu $t2, $zero, 0       # last_egg_broken = 0
addiu $t3, $zero, 0       # total_cost = 0
addiu $t4, $zero, 1       # low = 1
addiu $t5, $zero, 100     # high = building_height
addiu $t6, $zero, 0       # current_floor = 0
addiu $t7, $zero, 2       # eggs_remaining = 2
addiu $gp, $zero, 0       # up_floors = 0
addiu $sp, $zero, 0       # down_floors = 0

# 人力成本增长时期游戏循环
labor_loop:
# 检查循环条件
sltu $at, $t5, $t4        # high < low?
bne $at, $zero, labor_end
beq $t7, $zero, labor_end

# 计算mid = (low + high) / 2
addu $at, $t4, $t5
srl $at, $at, 1

# 计算上下楼
sltu $fp, $t6, $at        # current_floor < mid?
bne $fp, $zero, labor_go_up

# 下楼情况
subu $fp, $t6, $at
addu $sp, $sp, $fp
beq $zero, $zero, labor_update_pos
```

```
labor_go_up:
# 上楼情况
subu $fp, $at, $t6
addu $gp, $gp, $fp

labor_update_pos:
# current_floor = mid
addiu $t6, $at, 0

# total_drops++
addiu $t0, $t0, 1

# 检查鸡蛋是否摔破
sltu $fp, $s1, $at       # egg_durability < mid?
bne $fp, $zero, labor_egg_broken

# 鸡蛋没破
addiu $t2, $zero, 0      # last_egg_broken = 0
addiu $t4, $at, 1        # low = mid + 1
beq $zero, $zero, labor_continue

labor_egg_broken:
# 鸡蛋摔破
addiu $t1, $t1, 1        # total_eggs_broken++
addiu $t7, $t7, -1       # eggs_remaining--
addiu $t2, $zero, 1      # last_egg_broken = 1
addiu $t5, $at, -1       # high = mid - 1

labor_continue:
beq $zero, $zero, labor_loop

labor_end:
# 计算人力成本增长时期总成本
addiu $ra, $zero, 0      # temp_cost = 0

# 计算 up_floors * p1
```

```
addiu $fp, $zero, 0
labor_mult1_loop:
beq $gp, $zero, labor_mult1_done
addu $ra, $ra, $s5
addiu $gp, $gp, -1
beq $zero, $zero, labor_mult1_loop
labor_mult1_done:

# 计算 down_floors * p2
addiu $fp, $zero, 0
labor_mult2_loop:
beq $sp, $zero, labor_mult2_done
addu $ra, $ra, $s6
addiu $sp, $sp, -1
beq $zero, $zero, labor_mult2_loop
labor_mult2_done:

# 计算 eggs_broken * p3
addiu $fp, $zero, 0
labor_mult3_loop:
beq $t1, $zero, labor_mult3_done
addu $ra, $ra, $s7
addiu $t1, $t1, -1
beq $zero, $zero, labor_mult3_loop
labor_mult3_done:

# 保存人力成本增长时期结果
addiu $gp, $t0, 0        # total_drops -> $gp
addiu $sp, $t1, 0        # total_eggs_broken -> $sp
addiu $fp, $t2, 0        # last_egg_broken -> $fp
addiu $ra, $ra, 0        # total_cost -> $ra

# 程序结束
halt
```

## 二. 比萨塔摔鸡蛋游戏验证模型结果分析

根据比萨塔摔鸡蛋游戏的模拟结果，使用二分查找策略在 100 层建筑中测试鸡蛋耐摔值（42 层），得到以下计算结果：

### 6.2.1 游戏模拟参数

- 建筑高度：100 层
- 鸡蛋耐摔值：42 层
- 测试策略：二分查找算法
- 初始鸡蛋数：2 个

### 6.2.2 模拟过程详细分析

二分查找测试序列：

1. **测试 50 层**：鸡蛋摔破（$50 > 42$），上楼 50 层，摔破 1 个鸡蛋
2. **测试 25 层**：鸡蛋未破（$25 \leq 42$），下楼 25 层
3. **测试 37 层**：鸡蛋未破（$37 \leq 42$），上楼 12 层
4. **测试 43 层**：鸡蛋摔破（$43 > 42$），上楼 6 层，摔破 1 个鸡蛋

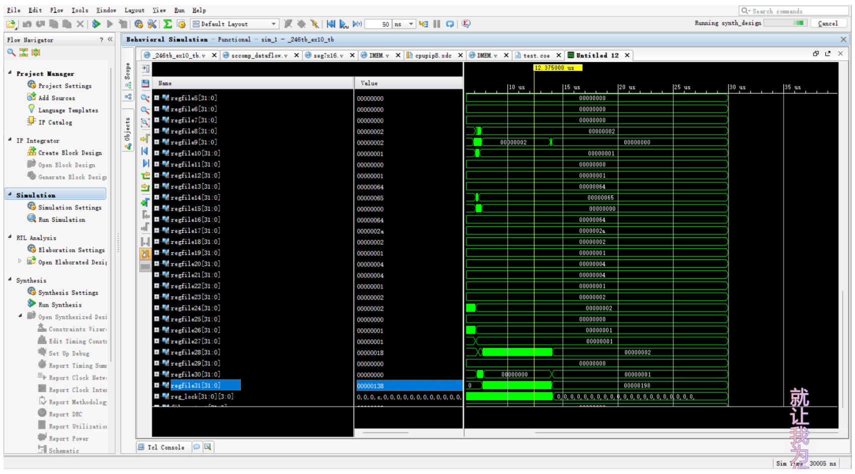
### 6.2.3 MIPS 寄存器存储结果

程序执行完毕后，结果存储在以下寄存器中：

| 时期 | 寄存器 | 数值和含义 |
|---|---|---|
| 物质匮乏时期 | \$20 | 4（摔的总次数） |
| | \$18 | 2（摔破的鸡蛋总数） |
| | \$19 | 1（最后鸡蛋状态：摔破） |
| | \$14 | 101 = 0x65（总成本 14 号寄存器值） |
| 人力成本增长时期 | \$28 | 4（摔的总次数） |
| | \$29 | 2（摔破的鸡蛋总数） |
| | \$30 | 1（最后鸡蛋状态：摔破） |
| | \$31 | 306 = 0x198（总成本 31 号寄存器值） |

### 6.2.4 成本分析结论

人力成本增长时期的总成本（301）显著高于物质匮乏时期（169），主要原因是上楼成本 $p_1$ 从 2 增加到 4，而鸡蛋成本 $p_3$ 的降低不足以抵消这一影响，反映了不同历史时期资源成本的变迁。



```
Behavioral Simulation - Functional - sim_1 - _246tb_ex10_tb

Name                 Value
regfile5[31:0]       00000000
regfile6[31:0]       00000000
regfile7[31:0]       00000000
regfile8[31:0]       00000002
regfile9[31:0]       00000002
regfile10[31:0]      00000001
regfile11[31:0]      00000000
regfile12[31:0]      00000001
regfile13[31:0]      00000064
regfile14[31:0]      00000065
regfile15[31:0]      00000000
regfile16[31:0]      00000064
regfile17[31:0]      0000002a
regfile18[31:0]      00000002
regfile19[31:0]      00000001
regfile20[31:0]      00000004
regfile21[31:0]      00000004
regfile22[31:0]      00000001
regfile23[31:0]      00000002
regfile24[31:0]      00000002
regfile25[31:0]      00000000
regfile26[31:0]      00000001
regfile27[31:0]      00000001
regfile28[31:0]      00000018
regfile29[31:0]      00000000
regfile30[31:0]      00000000
regfile31[31:0]      00000138
reg_lock[31:0][3:0]  0,0,0,c,0,0,0,0,0,0,0,0,0,0,0,0,
```

图 2:

## 第七部分　实验验算程序下板测试过程与实现

流水线 CPU 的 FPGA 下板测试遵循以下步骤完成：

1. 将 Verilog 设计文件综合后，通过 JTAG 下载至 FPGA 开发板
2. 配置测试向量生成模块与时钟源，设置仿真激励参数
3. 运行测试程序，通过逻辑分析仪监测 CPU 的输出结果与内部信号
4. 对比 FPGA 实测结果与 ModelSim 仿真结果，验证硬件功能的正确性

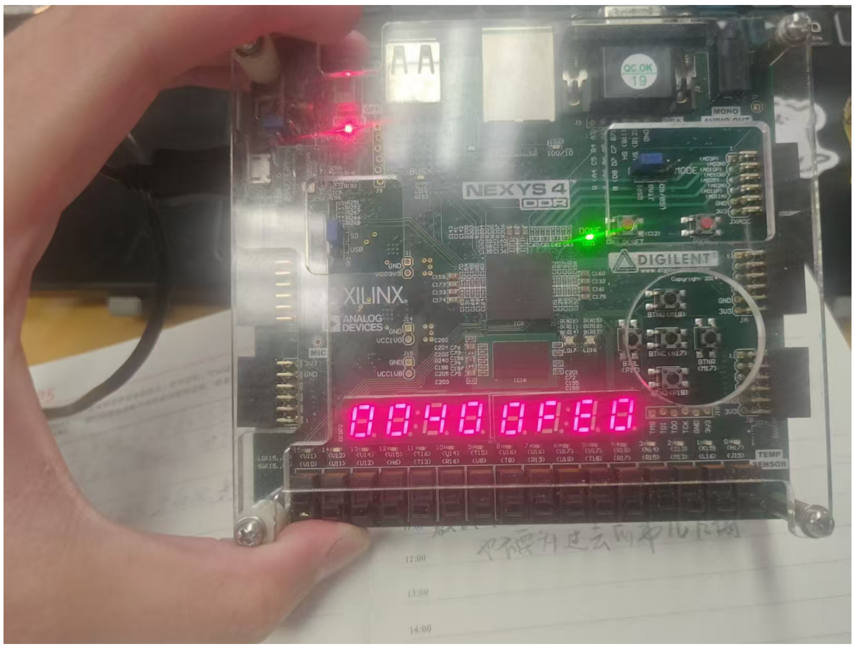


图 3:

# 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)

理想状态下，5 级流水线 CPU 的吞吐率为 1 条指令/时钟周期，即流水线稳定运行时，每个时钟周期可完成一条指令的执行。但在实际场景中，数据相关、控制相关与结构冲突会导致流水线停顿，使得实际吞吐率低于理想值。

### 8.1.2　加速比 (Speedup)

理论上，5 级流水线的理想加速比为 5，即相比单周期 CPU，5 级流水线的指令执行效率可提升 5 倍。但由于流水线冲突与停顿的存在，实际加速比会显著低于理论值。

### 8.1.3　效率 (Efficiency)

流水线效率的计算公式为：实际加速比 / 理论加速比，理想效率为 1。实际效率受数据相关、控制相关、分支预测准确性等因素影响，通常小于 1。

### 8.1.4　相关与冲突分析

流水线执行过程中的核心冲突类型及解决策略如下：

- **数据相关**：通过数据前递技术直接传递运算结果，无法前递时插入流水线气泡（停顿周期）
- **控制相关**：采用分支预测与延迟槽技术，减少分支指令带来的流水线冲刷
- **结构冲突**：通过增加硬件资源（如多端口存储器）或插入停顿周期，避免功能部件的资源竞争

### 8.1.5　CPU 的运行时间及存储器空间的使用

流水线 CPU 通过指令的并行执行，大幅缩短了指令的平均执行时间，提升了整体运行效率；在存储资源方面，流水线需要额外的流水寄存器暂存中间结果，虽增加了少量硬件开销，但整体资源占用处于可接受范围，是性能与资源的合理权衡。

# 第九部分　总结与体会

通过本次 MIPS 架构 5 级流水线 CPU 的设计与实现实验，我深入理解了流水线技术在提升 CPU 性能中的核心作用。将指令执行过程拆解为取指、译码、执行、访存、写

回五个阶段，通过并行处理多条指令，能够显著提高处理器的指令吞吐率。在设计过程中，针对数据相关、控制相关与结构冲突等问题，需采用数据前递、分支预测、流水线停顿等技术逐一解决，这也让我认识到流水线设计的复杂性与工程性。

本次实验成功实现了 ADD、ADDU、ADDI、ADDIU、SLL、LW、SW、SUBU、BNE、BEQ、SLTU、HALT 共 12 条指令的流水线执行，通过数据前递技术解决了大部分数据相关问题，借助分支预测机制降低了控制相关带来的性能损耗。

流水线技术虽能有效提升 CPU 的执行效率，但也引入了数据冒险、控制冒险与结构冒险等新问题。通过本次实验，我不仅掌握了流水线的基本原理与设计方法，还学会了分析与解决流水线中的各类冲突问题，对计算机系统结构的底层实现有了更深刻的认知。同时，FPGA 仿真与下板测试的过程，也让我体会到硬件设计从理论到实践的转化过程，提升了工程实践能力。

# 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)

```
`timescale 1ns / 1ps

`include "def.v"
module sccomp_dataflow (
    input clk,
    input reset,
    // alu
    output [31:0] a,
    output [31:0] b,
    output [3:0] aluc,
    output [31:0] aluo,
    output zero,
    output carry,
    output negative,
    output overflow,
    // bjudge
    output [31:0] rs,
    output [31:0] rt,
    input [31:0] instr,
    output [31:0] NPC_if_id,
    output B_PC_en,
    output [31:0] B_PC,
    //DMEM
    output [1:0] SC,
    output [2:0] LC,
    output [31:0] Data_in,
    output [31:0] DMEMaddr,
    output CS,
    output DM_W,
    output DM_R,
    input [31:0] Dataout,
    //EX_MEM
    output [3:0] doing_op_id_ex,
    output [31:0] instr_id_ex,
    output [31:0] aluo_ex_mem,
    output [31:0] b_ex_mem,
    output [31:0] instr_ex_mem,
    output [3:0] doing_op_ex_mem,


    //ID_EX
    output [31:0] instr_if_id,
    output [3:0] doing_op ,

    // IF_ID
    output [3:0] jpc_head,
    output [31:0] NPC,
    output [31:0] PC,
    output reg_detect_confict,
    output PC_bobl,
    output JPC_en,
    output [31:0] JPC,
    output halt,

    //IMEM

    //MEM_WB
```

```
output [4:0] rdc,
output [31:0] rdd,
output wen ,

//NPCmaker
output [31:0] NPC_out,

//regfile
output [4:0] rsc,
output [4:0] rtc,
output [31:0] rd,
output [31:0] regfile0,
output [31:0] regfile1,
output [31:0] regfile2,
output [31:0] regfile3,
output [31:0] regfile4,
output [31:0] regfile5,
output [31:0] regfile6,
output [31:0] regfile7,
output [31:0] regfile8,
output [31:0] regfile9,
output [31:0] regfile10,
output [31:0] regfile11,
output [31:0] regfile12,
output [31:0] regfile13,
output [31:0] regfile14,
output [31:0] regfile15,
output [31:0] regfile16,
output [31:0] regfile17,
output [31:0] regfile18,
output [31:0] regfile19,
output [31:0] regfile20,
output [31:0] regfile21,
output [31:0] regfile22,
output [31:0] regfile23,
output [31:0] regfile24,
output [31:0] regfile25,
    output [31:0] regfile26,
    output [31:0] regfile27,
    output [31:0] regfile28,
    output [31:0] regfile29,
    output [31:0] regfile30,
    output [31:0] regfile31

);
    cpu sccpu(
        .clk(clk),
        .reset(reset),
        .a(a),
        .b(b),
        .aluc(aluc),
        .aluo(aluo),
        .zero(zero),
        .carry(carry),
        .negative(negative),
        .overflow(overflow),
        .rs(rs),
        .rt(rt),
        .instr(instr),
        .NPC_if_id(NPC_if_id),
        .B_PC_en(B_PC_en),
        .B_PC(B_PC),
        .SC(SC),
        .LC(LC),
        .Data_in(Data_in),
        .DMEMaddr(DMEMaddr),
        .CS(CS),
        .DM_W(DM_W),
        .DM_R(DM_R),
        .Dataout(Dataout),
        .doing_op_id_ex(doing_op_id_ex),
        .instr_id_ex(instr_id_ex),
        .aluo_ex_mem(aluo_ex_mem),
        .b_ex_mem(b_ex_mem),
.instr_ex_mem(instr_ex_mem),
.doing_op_ex_mem(doing_op_ex_mem),
.instr_if_id(instr_if_id),
.doing_op(doing_op),
.jpc_head(jpc_head),
.NPC(NPC),
.PC(PC),
.reg_detect_confict(reg_detect_confict),
.PC_bobl(PC_bobl),
.JPC_en(JPC_en),
.JPC(JPC),
.halt(halt),
.rdc(rdc),
.rdd(rdd),
.wen(wen),
.NPC_out(NPC_out),
.rsc(rsc),
.rtc(rtc),
.rd(rd),
.regfile0(regfile0),
.regfile1(regfile1),
.regfile2(regfile2),
.regfile3(regfile3),
.regfile4(regfile4),
.regfile5(regfile5),
.regfile6(regfile6),
.regfile7(regfile7),
.regfile8(regfile8),
.regfile9(regfile9),
.regfile10(regfile10),
.regfile11(regfile11),
.regfile12(regfile12),
.regfile13(regfile13),
.regfile14(regfile14),
.regfile15(regfile15),
.regfile16(regfile16),
.regfile17(regfile17),
        .regfile18(regfile18),
        .regfile19(regfile19),
        .regfile20(regfile20),
        .regfile21(regfile21),
        .regfile22(regfile22),
        .regfile23(regfile23),
        .regfile24(regfile24),
        .regfile25(regfile25),
        .regfile26(regfile26),
        .regfile27(regfile27),
        .regfile28(regfile28),
        .regfile29(regfile29),
        .regfile30(regfile30),
        .regfile31(regfile31)
    );

    IMEM imem_inst(
        .address(PC),
        .instr_if_id(instr_if_id),
        .instr(instr)
    );

    DMEM dmem_inst(
        .clk(clk),
        .SC(SC),
        .LC(LC),
        .Data_in(Data_in),
        .DMEMaddr(DMEMaddr),
        .CS(CS),
        .DM_W(DM_W),
        .DM_R(DM_R),
        .Dataout(Dataout)
    );

endmodule
```

### 10.1.2 CPU 核心模块 (cpu.v)

```
`include "def.v"
`timescale 1ns / 1ps

module cpu (
    input clk,
    input reset,
    // alu
    output [31:0] a,
    output [31:0] b,
    output [3:0] aluc,
    output [31:0] aluo,
    output zero,
    output carry,
    output negative,
    output overflow,
    // bjudge
    output [31:0] rs,
    output [31:0] rt,
    input [31:0] instr,
    output [31:0] NPC_if_id,
    output B_PC_en,
    output [31:0] B_PC,
    //DMEM
    output [1:0] SC,
    output [2:0] LC,
    output [31:0] Data_in,
    output [31:0] DMEMaddr,
    output CS,
    output DM_W,
    output DM_R,
    input [31:0] Dataout,
    //EX_MEM
    output [3:0] doing_op_id_ex,
    output [31:0] instr_id_ex,
    output [31:0] aluo_ex_mem,
    output [31:0] b_ex_mem,
    output [31:0] instr_ex_mem,
    output [3:0] doing_op_ex_mem,


    //ID_EX
    output [31:0] instr_if_id,
    output [3:0] doing_op ,

    // IF_ID
    output [3:0] jpc_head,
    output [31:0] NPC,
    output [31:0] PC,
    output reg_detect_confict,
    output PC_bobl,
    output JPC_en,
    output [31:0] JPC,
    output halt,

    //IMEM

    //MEM_WB
    output [4:0] rdc,
    output [31:0] rdd,
    output wen ,

    //NPCmaker
    output [31:0] NPC_out,

    //regfile
    output [4:0] rsc,
    output [4:0] rtc,
    output [31:0] rd,
    output [31:0] regfile0,
    output [31:0] regfile1,
    output [31:0] regfile2,
    output [31:0] regfile3,
    output [31:0] regfile4,
        output [31:0] regfile5,
        output [31:0] regfile6,
        output [31:0] regfile7,
        output [31:0] regfile8,
        output [31:0] regfile9,
        output [31:0] regfile10,
        output [31:0] regfile11,
        output [31:0] regfile12,
        output [31:0] regfile13,
        output [31:0] regfile14,
        output [31:0] regfile15,
        output [31:0] regfile16,
        output [31:0] regfile17,
        output [31:0] regfile18,
        output [31:0] regfile19,
        output [31:0] regfile20,
        output [31:0] regfile21,
        output [31:0] regfile22,
        output [31:0] regfile23,
        output [31:0] regfile24,
        output [31:0] regfile25,
        output [31:0] regfile26,
        output [31:0] regfile27,
        output [31:0] regfile28,
        output [31:0] regfile29,
        output [31:0] regfile30,
        output [31:0] regfile31

    );
        alu alu_inst(
            .a(a),
            .b(b),
            .aluc(aluc),
            .r(aluo),
            .zero(zero),
            .carry(carry),
            .negative(negative),
            .overflow(overflow)
        );
        BJudge BJudge_inst(
            .rs(rs),
            .rt(rt),
            .instr(instr_if_id),
            .NPC_if_id(NPC_if_id),
            .B_PC_en(B_PC_en),
            .B_PC(B_PC)
        );

        EX_MEM EX_MEM_inst(
            .clk(clk),
            .reset(reset),
            .doing_op(doing_op_id_ex),
            .instr(instr_id_ex),
            .aluo(aluo),
            .b(b),
            .zero(zero),
            .carry(carry),
            .negative(negative),
            .overflow(overflow),
            .SC(SC),
            .LC(LC),
            .Data_in(Data_in),
            .DMEMaddr(DMEMaddr),
            .CS(CS),
            .DM_W(DM_W),
            .DM_R(DM_R),

            .aluo_ex_mem(aluo_ex_mem),
            .b_ex_mem(b_ex_mem),
            .instr_ex_mem(instr_ex_mem),
            .doing_op_ex_mem(doing_op_ex_mem)
        );

        ID_EX ID_EX_inst(
            .clk(clk),
            .reset(reset),
            .instr(instr_if_id),
            .rs(rs),
            .rt(rt),
            .doing_op(doing_op),
            .a(a),
            .b(b),
            .aluc(aluc),
            .instr_id_ex(instr_id_ex),
            .doing_op_id_ex(doing_op_id_ex)
        );

        IF_ID IF_ID_inst(
            .clk(clk),
            .reset(reset),
            .jpc_head(jpc_head),
            .NPC(NPC),
            .instr(instr),
            .PC(PC),
            .reg_detect_confict(reg_detect_confict),
            .PC_bobl(PC_bobl),
            .JPC_en(JPC_en),
            .JPC(JPC),
            .doing_op(doing_op),
            .instr_if_id(instr_if_id),
            .NPC_if_id(NPC_if_id),
            .halt(halt),
            .rsc(rsc),
            .rtc(rtc)
        );


        MEM_WB MEM_WB_inst(
            .clk(clk),
            .reset(reset),
            .doing_op(doing_op_ex_mem),
            .instr(instr_ex_mem),
            .ALUo(aluo_ex_mem),
            .Dataout(Dataout),
            .rdc(rdc),
            .rdd(rdd),
            .wen(wen)
        );

        NPCmaker NPCmaker_inst(
            .PC_bobl(PC_bobl),
            .detect_conflict(reg_detect_confict),
            .PC(PC),
            .NPC(NPC),
            .B_PC(B_PC),
            .B_PC_en(B_PC_en),
            .J_PC(JPC),
            .J_PC_en(JPC_en),
            .NPC_out(NPC_out)
        );

        PCreg PCreg_inst(
            .pc_clk(clk),
            .reset(reset),
            .npc_in(NPC_out),
            .halt(halt),
            .npc(NPC),
            .pc(PC),
            .jpc_head(jpc_head)
        );
        regfile cpu_ref(
            .clk(clk),
            .reset(reset),
            .wen(wen),
            .rdc(rdc),
            .rdd(rdd),
            .rsc(rsc),
            .rtc(rtc),
        .doing_op(doing_op),
        .instr(instr_if_id),
        .ALUo_EX(aluo),
        .ALUo_MEM(aluo_ex_mem),
        .ALUo_WB(rdd),
        .Data_MEM(Dataout),
        .Data_WB(rdd),
        .rs(rs),
        .rt(rt),
        .rd(rd),
        .detect_conflict(reg_detect_confict),
        .regfile0(regfile0),
        .regfile1(regfile1),
        .regfile2(regfile2),
        .regfile3(regfile3),
        .regfile4(regfile4),
        .regfile5(regfile5),
        .regfile6(regfile6),
        .regfile7(regfile7),
        .regfile8(regfile8),
        .regfile9(regfile9),
        .regfile10(regfile10),
        .regfile11(regfile11),
        .regfile12(regfile12),
        .regfile13(regfile13),
        .regfile14(regfile14),
        .regfile15(regfile15),
        .regfile16(regfile16),
        .regfile17(regfile17),
        .regfile18(regfile18),
        .regfile19(regfile19),
        .regfile20(regfile20),
        .regfile21(regfile21),
        .regfile22(regfile22),
        .regfile23(regfile23),
        .regfile24(regfile24),
        .regfile25(regfile25),
                .regfile26(regfile26),
                .regfile27(regfile27),
                .regfile28(regfile28),
                .regfile29(regfile29),
                .regfile30(regfile30),
                .regfile31(regfile31)
            );



        endmodule
```

### 10.1.3 定义文件 (def.v)

```
`define r_op           6'b000000
`define sll_func       6'b000000
`define srl_func       6'b000010
`define sra_func       6'b000011
`define sllv_func      6'b000100
`define srlv_func      6'b000110
`define srav_func      6'b000111
`define jr_func        6'b001000
`define jalr_func      6'b001001
`define mfhi_func      6'b010000
`define mthi_func      6'b010001
`define mflo_func      6'b010010
`define mtlo_func      6'b010011
`define mult_func      6'b011000
`define multu_func     6'b011001
`define div_func       6'b011010
`define divu_func      6'b011011
`define add_func       6'b100000
`define clz_func       6'b100000
`define addu_func      6'b100001
`define sub_func       6'b100010
`define subu_func      6'b100011
`define and_func       6'b100100
```

```
`define or_func       6'b100101
`define xor_func      6'b100110
`define nor_func      6'b100111
`define slt_func      6'b101010
`define sltu_func     6'b101011
`define teq_func      6'b110100


`define bgez_op       6'b000001
`define j_op          6'b000010
`define jal_op        6'b000011
`define beq_op        6'b000100
`define bne_op        6'b000101
`define addi_op       6'b001000
`define addiu_op      6'b001001
`define slti_op       6'b001010
`define sltiu_op      6'b001011
`define andi_op       6'b001100
`define ori_op        6'b001101
`define xori_op       6'b001110
`define lui_op        6'b001111
`define mfc0_op       6'b010000
`define mtc0_op       6'b010000
`define clz_op        6'b011100
`define lb_op         6'b100000
`define lh_op         6'b100001
`define lw_op         6'b100011
`define lbu_op        6'b100100
`define lhu_op        6'b100101
`define sb_op         6'b101000
`define sh_op         6'b101001
`define sw_op         6'b101011

`define bgez_rt       5'b00001

`define mfc0_rs       5'b00000
`define mtc0_rs       5'b00100
```

```
`define break_instr   32'b000000_00000_00000_00000_00000_001101
`define syscall_instr 32'b000000_00000_00000_00000_00000_001100
`define eret_instr    32'b010000_10000_00000_00000_00000_011000
`define halt_instr    32'b111111_11111_11111_11111_11111_111111
`define nop_instr     32'b000000_00000_00000_00000_00000_000000

`define add_aluc      4'b0010
`define addu_aluc     4'b0000
`define sub_aluc      4'b0011
`define subu_aluc     4'b0001
`define and_aluc      4'b0100
`define or_aluc       4'b0101
`define xor_aluc      4'b0110
`define nor_aluc      4'b0111
`define slt_aluc      4'b1011
`define sltu_aluc     4'b1010
`define sll_aluc      4'b1110
`define srl_aluc      4'b1101
`define sra_aluc      4'b1100
`define lui_aluc      4'b1000
`define bgez_aluc     4'b1001

`define nvl_alumctr   3'b000
`define mult_alumctr  3'b001
`define multu_alumctr 3'b010
`define div_alumctr   3'b011
`define divu_alumctr  3'b100
`define mthi_alumctr  3'b101
`define mtlo_alumctr  3'b110

`define SYSCALL_cause 4'b1000
`define BREAK_cause   4'b1001
`define TEQ_cause     4'b1101

`define sb_dmem 2'b10
`define sh_dmem 2'b01
```

```
`define sw_dmem 2'b00
`define lw_dmem 3'b000
`define lhu_dmem 3'b001
`define lh_dmem  3'b010
`define lb_dmem  3'b100
`define lbu_dmem 3'b011

//下面定义关于对应op在doing_op当中对应的宏

`define add 1
`define addu 2
`define addi 3
`define addiu 4
`define sll 5
`define halt 6
`define lw 7
`define sw 8
`define subu 9
`define bne 10
`define beq 11
`define sltu 12
```

### 10.1.4 算术逻辑单元 (alu.v)

```
`timescale 1ns / 1ps
module alu(
input [31:0] a,   //32 位输入, 操作数1
input [31:0] b,   //32 位输入, 操作数2
input [3:0] aluc, //4位输入, 控制 alu 的操作
output reg [31:0] r, //32 位输出, 由a、b经过aluc指定的操作生成
output reg zero,
output reg carry,
//0 标志位
// 进位标志位
output reg negative,   // 负数标志位
output reg overflow   // 溢出标志位
);
  always @ (*) begin
```

```
case (aluc)
  4'b0000://无符号加法
  begin
    r <= a + b;
    zero <= (r == 0);
    carry <= (a[31] & b[31]) | (a[31] & ~r[31]) | (b[31] & ~r[31]);
    negative <= (r[31] == 1);
    overflow <= 0;
  end
  4'b0010://有符号加法
  begin
    r <= $signed(a) + $signed(b);
    zero <= (r == 0);
    carry <= 0;
    negative <= (r[31] == 1);
    overflow <= (a[31] & b[31] & ~r[31]) | (~a[31] & ~b[31] & r[31]);
  end
  4'b0001://无符号减法
  begin
    r <= a - b;
    zero <= (r == 0);
    carry <= (~a[31] & b[31]) | (~a[31] & r[31]) | (~b[31] & r[31]);
    negative <= (r[31] == 1);
    overflow <= 0;
  end
  4'b0011://有符号减法
  begin
    r <= $signed(a) - $signed(b);
    zero <= (r == 0);
    carry <= 0;
    negative <= (r[31] == 1);
    overflow <= (~a[31] & b[31] & r[31]) | (a[31] & ~b[31] & ~r[31]);
  end
  4'b0100://与运算
  begin
    r <= a & b;
    zero <= (r == 0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b0101://或运算
begin
  r <= a | b;
  zero <= (r == 0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b0110://异或运算
begin
  r <= a ^ b;
  zero <= (r == 0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b0111://nor运算
begin
  r <= ~(a | b);
  zero <= (r == 0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b1000://Lui运算
begin
  r={b[15:0],16'b0};
  zero <= (r==0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b1001://bgez运算
```

```
begin
  r=a;
  zero <= (r == 0);
  carry <= 0;
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b1011://Slt运算
begin
  r <= ($signed(a) < $signed(b));
  zero <= (($signed(a) - $signed(b)) == 0);
  carry <= 0;
  negative <= ($signed(a) < $signed(b));
  overflow <= 0;
end
4'b1010://Sltu运算
begin
  r <= (a < b);
  zero <= ((a-b) == 0);
  carry <= (a<b);
  negative <= (r[31] == 1);
  overflow <= 0;
end
4'b1100://Sra运算
begin
  r=($signed(b) >>> $signed(a));
  zero <= (r == 0);
  //carry为最后一次被移出的位的数值
  if(a<32&&a>0)
      carry <= b[a];
  else if(a==0)
      carry <= 0;
  else
      carry=b[31];
  negative <= (r[31] == 1);
  overflow <= 0;
end
```

```
      4'b1101://srl
      begin
        r=b>>a;
        zero <= (r == 0);
        //carry为最后一次被移出的位的数值
        if(a<32&&a>0)
            carry <= b[a];
        else
            carry <= 0;
        negative <= (r[31] == 1);
        overflow <= 0;
      end
      4'b1110://sll
      begin
        r=b<<a;
        zero <= (r == 0);
        //carry为最后一次被移出的位的数值
        if(a<32&&a>0)
            carry=b[32-a];
        else
            carry=0;
        negative <= (r[31] == 1);
        overflow <= 0;
      end
      4'b1111://sla
      begin
        r=b<<a;
        zero <= (r == 0);
        //carry为最后一次被移出的位的数值
        carry <= b[31];
        negative <= (r[31] == 1);
        overflow <= 0;
      end
    endcase
    end
endmodule
```

### 10.1.5 分支判断模块 (BJudge.v)

```
`include "def.v"
`timescale 1ns / 1ps

module BJudge(
    input [31:0] rs,
    input [31:0] rt,
    input [31:0] instr,
    input [31:0] NPC_if_id,
    output B_PC_en,
    output [31:0] B_PC
);

assign B_PC=(B_PC_en)?
    NPC_if_id+{{14{instr[15]}},instr[15:0],2'b00}:
    NPC_if_id;

assign B_PC_en=(instr[31:26]==`beq_op)?(rs==rt):
                (instr[31:26]==`bne_op)?(rs!=rt):
                0;

endmodule
```

### 10.1.6 PC 寄存器模块 (PCreg.v)

```
`timescale 1ns / 1ps

module PCreg(
    input pc_clk,
    input reset,           // 低电平有效复位
    input [31:0] npc_in,
    input halt,
    output [31:0] npc,
    output reg [31:0] pc,
    output [3:0] jpc_head
);
```

```
assign jpc_head = pc[31:28];  // 取PC高4位

// 异步复位，上升沿触发
reg Halting;
always @(posedge pc_clk) begin
    if (reset) begin
        Halting <= 1'b0;
    end else if (halt) begin
        Halting <= 1'b1;
    end
end



always @(posedge pc_clk) begin
    if (reset) begin       // 复位时PC清零
        pc <= 32'b0;
    end else if (Halting||halt) begin
        pc <= pc;
    end else begin
        pc <= npc_in;
    end
end

assign npc = pc + 32'd4;  // 计算下一条指令地址

endmodule
```

### 10.1.7 数据存储器 (DMEM.v)

```
//// 被注释掉的是前仿真的版本，会出现在后仿真的过程中synthesis过慢的问题，原因是过大的
`timescale 1ns / 1ps
`include "def.v"

module DMEM(
    input clk,
    input [1:0]SC,
    input [2:0]LC,
    input [31:0] Data_in,
    input [31:0] DMEMaddr,
    input CS,
    input DM_W,
    input DM_R,
    output [31:0] Dataout
);

wire [7:0] dmem1_w;
wire [7:0] dmem2_w;
wire [7:0] dmem3_w;
wire [7:0] dmem4_w;
wire [7:0] dmem1_r;
wire [7:0] dmem2_r;
wire [7:0] dmem3_r;
wire [7:0] dmem4_r;

wire we1;
wire we2;
wire we3;
wire we4;



assign dmem1_w = Data_in[7:0];
assign dmem2_w = Data_in[15:8];
assign dmem3_w = Data_in[23:16];
assign dmem4_w = Data_in[31:24];

assign we1 = (SC == `sw_dmem || SC == `sh_dmem || SC == `sb_dmem) && DM_W && CS;
assign we2 = (SC == `sw_dmem || SC == `sh_dmem) && DM_W && CS;
assign we3 = (SC == `sw_dmem) && DM_W && CS;
assign we4 = (SC == `sw_dmem) && DM_W && CS;

assign Dataout = (CS && DM_R) ? (LC == `lw_dmem) ? {dmem4_r, dmem3_r, dmem2_r, dmem1_
                                (LC == `lhu_dmem) ? {16'b0, dmem2_r, dmem1_r} :
                                (LC == `lh_dmem)  ? {{16{dmem2_r[7]}}, dmem2_r, dmem1
                                (LC == `lb_dmem)  ? {{24{dmem1_r[7]}}, dmem1_r} :
```

```
                                                (LC == `lbu_dmem) ? {24'b0, dmem1_r} : 32'bz : 32'bz;

dmem1 dmem1_uut(
    .a(DMEMaddr[10:0]),
    .d(dmem1_w),
    .clk(clk),
    .we(we1),
    .spo(dmem1_r)
);

dmem1 dmem2_uut(
    .a(DMEMaddr[10:0]),
    .d(dmem2_w),
    .clk(clk),
    .we(we2),
    .spo(dmem2_r)
);

dmem1 dmem3_uut(
    .a(DMEMaddr[10:0]),
    .d(dmem3_w),
    .clk(clk),
    .we(we3),
    .spo(dmem3_r)
);

dmem1 dmem4_uut(
    .a(DMEMaddr[10:0]),
    .d(dmem4_w),
    .clk(clk),
    .we(we4),
    .spo(dmem4_r)
);


endmodule
```

### 10.1.8 流水级间寄存器 (EX_MEM.v)

```
`timescale 1ns / 1ps

`include "def.v"
module EX_MEM(
    input clk,
    input reset,
    input [3:0] doing_op,
    input [31:0] instr,

    input [31:0] aluo,
    input [31:0] b,

    input zero,
    input carry,
    //0 标志位
    // 进位标志位
    input negative,   // 负数标志位
    input overflow,   // 溢出标志位
    output [1:0]SC,
    output [2:0]LC,
    output [31:0] Data_in,
    output [31:0] DMEMaddr,
    output CS,
    output DM_W,
    output DM_R,

    output [31:0] aluo_ex_mem,
    output [31:0] b_ex_mem,
    output [31:0] instr_ex_mem,
    output [3:0] doing_op_ex_mem
);
    reg zero_r, carry_r, negative_r, overflow_r;
    always @(posedge clk) begin
        if (reset) begin
            zero_r <= 0;
            carry_r <= 0;
            negative_r <= 0;
            overflow_r <= 0;
        end else begin
            zero_r <= zero;
            carry_r <= carry;
            negative_r <= negative;
            overflow_r <= overflow;
        end
    end
    reg [31:0] aluo_r , b_r , instr_r;
    reg [3:0] doing_op_r;

    always @ (posedge clk) begin
        if (reset) begin
            aluo_r <= 0;
            b_r <= 0;
            doing_op_r <= 0;
            instr_r <= 0;
        end else begin
            aluo_r<=aluo;
            b_r<=b;
            doing_op_r<=doing_op;
            instr_r<=instr;
        end
    end

    assign aluo_ex_mem = aluo_r;
    assign b_ex_mem = b_r;
    assign instr_ex_mem = instr_r;
    assign doing_op_ex_mem = doing_op_r;
    assign SC = (doing_op_r==`sw) ? `sw_dmem:2'b0;
    assign LC = (doing_op_r==`lw) ? `lw_dmem:2'b0;

    assign CS= (doing_op_r==`sw) || (doing_op_r==`lw) ? 1'b1:1'b0;
    assign DM_W = (doing_op_r==`sw) ? 1'b1:1'b0;
    assign DM_R = (doing_op_r==`lw) ? 1'b1:1'b0;
    assign DMEMaddr = (doing_op_r==`sw || doing_op_r==`lw) ? aluo_r:32'b0;
```

```
    assign Data_in = (doing_op_r==`sw) ? b_r:32'b0;

endmodule
```

### 10.1.9 流水级间寄存器 (ID_EX.v)

```
`timescale 1ns / 1ps

`include "def.v"
module ID_EX(
    input clk,
    input reset,
    input [31:0] instr,
    input [31:0] rs,
    input [31:0] rt,
    input [3:0] doing_op,
    output [31:0] a,
    output [31:0] b,
    output [3:0] aluc,
    output [31:0] instr_id_ex,
    output [3:0] doing_op_id_ex
);


reg [31:0] instr_reg, rs_reg, rt_reg;
reg [3:0] doing_op_reg;

assign instr_id_ex = instr_reg;
assign doing_op_id_ex = doing_op_reg;

always @(posedge clk) begin
    if (reset) begin
        instr_reg <= 0;
        rs_reg <= 0;
        rt_reg <= 0;
        doing_op_reg <= 0;
    end else begin
        instr_reg<=instr;
```

```
            rs_reg<=rs;
            rt_reg<=rt;
            doing_op_reg <= doing_op;
        end
    end

    assign a =
        (doing_op_reg==`add) ? rs_reg:
        (doing_op_reg==`addu) ? rs_reg:
        (doing_op_reg==`addi) ? rs_reg:
        (doing_op_reg==`addiu) ? rs_reg:
        (doing_op_reg==`subu) ? rs_reg:
        (doing_op_reg==`sltu) ? rs_reg:
        (doing_op_reg==`lw) ? rs_reg:
        (doing_op_reg==`sw) ? rs_reg:
        (doing_op_reg==`sll) ? {27'b0,instr_reg[10:6]}://shamt
        (doing_op_reg==`beq) ? rs_reg:
        (doing_op_reg==`bne) ? rs_reg:
        32'b0;
    assign b =
        (doing_op_reg==`add) ? rt_reg:
        (doing_op_reg==`addu) ? rt_reg:
        (doing_op_reg==`addi) ? {{16{instr_reg[15]}},instr_reg[15:0]}: // sign extend imd
        (doing_op_reg==`addiu) ? {{16{instr_reg[15]}},instr_reg[15:0]}: // sign extend im
        (doing_op_reg==`subu) ? rt_reg:
        (doing_op_reg==`sltu) ? rt_reg:
        (doing_op_reg==`lw) ? {{16{instr_reg[15]}},instr_reg[15:0]}: // sign extend imdt
        (doing_op_reg==`sw) ? {{16{instr_reg[15]}},instr_reg[15:0]}: // sign extend imdt
        (doing_op_reg==`sll) ? rt_reg:
        (doing_op_reg==`beq) ? rt_reg:
        (doing_op_reg==`bne) ? rt_reg:
        32'b0;
    assign aluc =
        (doing_op_reg==`add) ? `add_aluc:
        (doing_op_reg==`addu) ? `addu_aluc:
        (doing_op_reg==`addi) ? `add_aluc:
        (doing_op_reg==`addiu) ? `addu_aluc:
```

```
    (doing_op_reg==`subu) ? `subu_aluc:
    (doing_op_reg==`sltu) ? `sltu_aluc:
    (doing_op_reg==`lw) ? `add_aluc:
    (doing_op_reg==`sw) ? `add_aluc:
    (doing_op_reg==`sll) ? `sll_aluc:
    (doing_op_reg==`beq) ? `sub_aluc:
    (doing_op_reg==`bne) ? `sub_aluc:
    4'b0;



endmodule
```

### 10.1.10 流水级间寄存器 (IF_ID.v)

```
`timescale 1ns / 1ps
`include "def.v"
module IF_ID(
    input clk,
    input reset,
    input [3:0] jpc_head,
    input [31:0] NPC,
    input [31:0] instr,
    input [31:0] PC,
    input reg_detect_confict,
    output [4:0] rsc,
    output [4:0] rtc,
    output PC_bobl,
    output JPC_en,
    output [31:0] JPC,
    output [3:0] doing_op,
    output [31:0] instr_if_id,
    output [31:0] NPC_if_id,
    output halt

);
```

```
reg [31:0] NPC_reg, instr_reg;
reg [3:0] doing_op_last;

always @(posedge clk)begin
    if (reset) begin
        NPC_reg <= 0;
        instr_reg<=0;
    end else if(reg_detect_confict) begin
        NPC_reg<=NPC_reg;
        instr_reg<=instr_reg;
    end else begin
        NPC_reg <= NPC;
        instr_reg<=instr;
    end
end
assign halt=(instr==`halt_instr);
assign instr_if_id=instr_reg;

assign NPC_if_id=NPC_reg;
//遇到跳转指令，直接跳转
assign JPC_en=0;
assign JPC={jpc_head, instr[25:21],1'b0,instr[19:0],2'b0};

assign PC_bobl=(instr[31:26]==`beq_op||
                instr[31:26]==`bne_op);
// always @(posedge clk) begin
//     if (reset) begin
//         doing_op <= 0;
//     end else if(reg_detect_confict) begin
//         doing_op<=doing_op;
//     end else if (instr== `halt_instr) begin
//         doing_op <= `halt;
//     end else begin
//         case (instr[31:26])
//             `lw_op: doing_op <= `lw;
//             `sw_op: doing_op <= `sw;
//             `beq_op: doing_op <= `beq;
//             `bne_op: doing_op <= `bne;
//             `addi_op: doing_op <= `addi;
//             `addiu_op: doing_op <= `addiu;

//             `r_op: begin
//                 case (instr[5:0])
//                     `add_func: doing_op <= `add;
//                     `addu_func: doing_op <= `addu;
//                     `subu_func: doing_op <= `subu;
//                     `sll_func: doing_op <= `sll;
//                     `sltu_func: doing_op <= `sltu;
//                     default: doing_op <= 0;
//                 endcase
//             end
//             default: doing_op <= 0;
//         endcase
//     end
// end
always @(posedge clk) begin
    if (reset) begin
        doing_op_last <= 0;
    end else begin
        doing_op_last <= doing_op;
    end
end
assign doing_op= (reg_detect_confict ) ? doing_op_last :
                (instr_reg== `halt_instr) ? `halt :
                (instr_reg[31:26]==`lw_op) ? `lw :
                (instr_reg[31:26]==`sw_op) ? `sw :
                (instr_reg[31:26]==`beq_op) ? `beq :
                (instr_reg[31:26]==`bne_op) ? `bne :
                (instr_reg[31:26]==`addi_op) ? `addi :
                (instr_reg[31:26]==`addiu_op) ? `addiu :
                (instr_reg[31:26]==`r_op) ?
                    (instr_reg[5:0]==`add_func) ? `add :
                    (instr_reg[5:0]==`addu_func) ? `addu :
                    (instr_reg[5:0]==`subu_func) ? `subu :
                        (instr_reg[5:0]==`sll_func) ? `sll :
                        (instr_reg[5:0]==`sltu_func) ? `sltu
                        : 0
                    : 0;
    assign rsc= instr_if_id[25:21];
    assign rtc= instr_if_id[20:16];
endmodule
```

### 10.1.11 指令存储器 (IMEM.v)

```
`timescale 1ns / 1ps

`include "def.v"
module IMEM(
    input [31:0] address,
    input [31:0] instr_if_id,

    output [31:0] instr
);
    wire [31:0] instrT;
    wire PC_bobl;
    assign PC_bobl=(instr_if_id[31:26]==`beq_op||
                instr_if_id[31:26]==`bne_op);
    assign instr = PC_bobl?`nop_instr:instrT;

      imem imem_ip(
          .a(address[12:2]),
          .spo(instrT)
      );
//   reg [31:0] IMEMreg [0:2047];
//   assign instrT=IMEMreg[address[12:2]];

//   initial begin
////       $readmemh("E:/Homeworks/cpupip8/testdata/1_addi.hex.txt", IMEMreg);
////       $readmemh("E:/Homeworks/cpupip8/testdata/2_addiu.hex.txt", IMEMreg);
//      $readmemh("E:/Homeworks/cpupip8/testdata/9_addu.hex.txt", IMEMreg);
////      $readmemh("E:/Homeworks/cpupip8/testdata/11_beq.hex.txt", IMEMreg);
////       $readmemh("E:/Homeworks/cpupip8/testdata/12_bne.hex.txt", IMEMreg);
```

```
////        $readmemh("E:/Homeworks/cpupip8/testdata/16.26_lwsw.hex.txt",IMEMreg);
////        $readmemh("E:/Homeworks/cpupip8/testdata/16.26_lwsw2.hex.txt",IMEMreg);
////        $readmemh("E:/Homeworks/cpupip8/testdata/20_sll.hex.txt", IMEMreg);
////        $readmemh("E:/Homeworks/cpupip8/testdata/22_sltu.hex.txt", IMEMreg);
////        $readmemh("E:/Homeworks/cpupip8/testdata/25_subu.hex.txt", IMEMreg);

//    end


endmodule
```

### 10.1.12 流水级间寄存器 (MEM_WB.v)

```
`timescale 1ns / 1ps

`include "def.v"
module MEM_WB (
    input clk,
    input reset,
    input [3:0] doing_op,
    input [31:0] instr,

    input [31:0]ALUo,
    input [31:0] Dataout,

    output [4:0] rdc,
    output [31:0] rdd,
    output wen

);



    reg [31:0] doing_op_r, instr_r , ALUo_r, Dataout_r;
    always @(posedge clk) begin
        if (reset) begin
            doing_op_r <= 0;
            instr_r <= 0;
            ALUo_r <= 0;
            Dataout_r <= 0;
        end else begin
            doing_op_r<=doing_op;
            instr_r<=instr;
            ALUo_r <= ALUo;
            Dataout_r <= Dataout;
        end
    end
    assign wen = (
        doing_op_r == `add||
        doing_op_r == `addu||
        doing_op_r == `addi||
        doing_op_r == `addiu||
        doing_op_r == `subu||
        doing_op_r == `sll||
        doing_op_r == `lw ||
        doing_op_r == `sltu
    );
    assign rdc = (
        doing_op_r == `add ||
        doing_op_r == `addu ||
        doing_op_r == `subu ||
        doing_op_r == `sll ||
        doing_op_r == `sltu
    ) ? instr_r[15:11]:
    (
        doing_op_r == `addi||
        doing_op_r == `addiu||
        doing_op_r == `lw
    ) ? instr_r[20:16]:
     5'b0;
    assign rdd = (
        doing_op_r == `add ||
        doing_op_r == `addu ||
        doing_op_r == `subu ||
        doing_op_r == `sll ||
        doing_op_r == `sltu ||
            doing_op_r == `addi ||
            doing_op_r == `addiu
        ) ? ALUo_r:
        (
            doing_op_r == `lw
        )? Dataout_r:
        32'b0;


    endmodule
```

### 10.1.13 下一 PC 生成模块 (NPCmaker.v)

```
`timescale 1ns / 1ps

module NPCmaker(
    input PC_bobl,
    input detect_conflict,
    input [31:0]PC,
    input [31:0]NPC,
    input [31:0]B_PC,
    input B_PC_en,
    input [31:0]J_PC,
    input J_PC_en,
    output [31:0]NPC_out
);
    assign NPC_out =(PC_bobl||detect_conflict)? PC :
                    (J_PC_en)? J_PC :
                    (B_PC_en)? B_PC :
                    NPC;
endmodule
```

### 10.1.14 寄存器文件 (regfile.v)

```
`timescale 1ns / 1ps

`include "def.v"
module regfile(
    input clk,
input reset,
input wen,
input [4:0] rdc,
input [31:0] rdd,
input [4:0]rsc,
input [4:0]rtc,
input [3:0] doing_op,
input [31:0] instr,
input [31:0] ALUo_EX,
input [31:0] ALUo_MEM,
input [31:0] ALUo_WB,
input [31:0] Data_MEM,
input [31:0] Data_WB,
output [31:0] rt,
output [31:0] rd,
output [31:0] rs,
output detect_conflict,
output [31:0] regfile0,
output [31:0] regfile1,
output [31:0] regfile2,
output [31:0] regfile3,
output [31:0] regfile4,
output [31:0] regfile5,
output [31:0] regfile6,
output [31:0] regfile7,
output [31:0] regfile8,
output [31:0] regfile9,
output [31:0] regfile10,
output [31:0] regfile11,
output [31:0] regfile12,
output [31:0] regfile13,
output [31:0] regfile14,
output [31:0] regfile15,
output [31:0] regfile16,
output [31:0] regfile17,
output [31:0] regfile18,
output [31:0] regfile19,
    output [31:0] regfile20,
    output [31:0] regfile21,
    output [31:0] regfile22,
    output [31:0] regfile23,
    output [31:0] regfile24,
    output [31:0] regfile25,
    output [31:0] regfile26,
    output [31:0] regfile27,
    output [31:0] regfile28,
    output [31:0] regfile29,
    output [31:0] regfile30,
    output [31:0] regfile31
);

    reg [31:0] array_reg[31:0];
    reg [3:0] reg_lock[31:0];

    assign regfile0 = array_reg[0];
    assign regfile1 = array_reg[1];
    assign regfile2 = array_reg[2];
    assign regfile3 = array_reg[3];
    assign regfile4 = array_reg[4];
    assign regfile5 = array_reg[5];
    assign regfile6 = array_reg[6];
    assign regfile7 = array_reg[7];
    assign regfile8 = array_reg[8];
    assign regfile9 = array_reg[9];
    assign regfile10 = array_reg[10];
    assign regfile11 = array_reg[11];
    assign regfile12 = array_reg[12];
    assign regfile13 = array_reg[13];
    assign regfile14 = array_reg[14];
    assign regfile15 = array_reg[15];
    assign regfile16 = array_reg[16];
    assign regfile17 = array_reg[17];
    assign regfile18 = array_reg[18];
    assign regfile19 = array_reg[19];
    assign regfile20 = array_reg[20];
    assign regfile21 = array_reg[21];
    assign regfile22 = array_reg[22];
    assign regfile23 = array_reg[23];
    assign regfile24 = array_reg[24];
    assign regfile25 = array_reg[25];
    assign regfile26 = array_reg[26];
    assign regfile27 = array_reg[27];
    assign regfile28 = array_reg[28];
    assign regfile29 = array_reg[29];
    assign regfile30 = array_reg[30];
    assign regfile31 = array_reg[31];

    integer i;  // 循环变量

    always @(posedge clk) begin
        if (reset) begin
            for (i = 0; i < 32; i=i+1) begin
                array_reg[i] <= 32'h0;  // 寄存器复位清零
            end
        end else if(wen && rdc != 5'b0) begin  // 写使能且写入地址不为0（寄存器0恒为0
            for (i = 0; i < 32; i=i+1) begin
                if (rdc == i) begin
                    array_reg[i] <= rdd;  // 写入指定寄存器
                end else begin
                    array_reg[i] <= array_reg[i];  // 其他寄存器保持不变
                end
            end
        end else begin
            for (i=0;i<32;i=i+1) begin
                array_reg[i] = array_reg[i];
            end
        end
    end

    // 寄存器值读出逻辑
```

```
wire lock_en;
// assign lock_en=(doing_op==`add
//      ||doing_op==`addu
//      ||doing_op==`addi
//      ||doing_op==`addiu
//      ||doing_op==`sll
//      ||doing_op==`lw
//      ||doing_op==`subu
//      ||doing_op==`sltu);
wire [4:0] rdc_to_lock;
assign rdc_to_lock=(doing_op==`add ||
                    doing_op==`addu ||
                    doing_op==`sll ||
                    doing_op==`subu||doing_op==`sltu)? instr[15:11]:
                    (doing_op==`lw||doing_op==`addi||doing_op==`addiu)?instr[20:1
assign lock_en= (rdc_to_lock!=5'b0);
always @(posedge clk) begin
    if (reset) begin
        for(i=0;i<32;i=i+1)begin
            reg_lock[i]<=4'b0;
        end
    end else begin
        for(i=0;i<32;i=i+1) begin
            if(lock_en && rdc_to_lock==i) begin
                reg_lock[i]<=(doing_op==`add ||
                    doing_op==`addu ||
                    doing_op==`sll ||
                    doing_op==`subu||doing_op==`sltu||doing_op==`addi||doing_op==
                    (doing_op==`lw)? 4'b1010 :4'b0;
            end else begin
                if(reg_lock[i][1:0]!=2'b0) begin
                    reg_lock[i]<=reg_lock[i]-1;
                end
                else begin
```

```
                    reg_lock[i]<=4'b0;
                end
            end
        end
    end
end
assign rs = (reg_lock[rsc]==0)?array_reg[rsc]:
            (reg_lock[rsc]==4'b1110)?ALUo_EX:
            (reg_lock[rsc]==4'b1101)?ALUo_MEM:
            (reg_lock[rsc]==4'b1100)?rdd:
            (reg_lock[rsc]==4'b1001)?Data_MEM:
            (reg_lock[rsc]==4'b1000)?rdd:array_reg[rsc];
assign rt = (reg_lock[rtc]==0)?array_reg[rtc]:
            (reg_lock[rtc]==4'b1110)?ALUo_EX:
            (reg_lock[rtc]==4'b1101)?ALUo_MEM:
            (reg_lock[rtc]==4'b1100)?rdd:
            (reg_lock[rtc]==4'b1001)?Data_MEM:
            (reg_lock[rtc]==4'b1000)?rdd:array_reg[rtc];
assign rd = (reg_lock[rdc]==0)?array_reg[rdc]:
            (reg_lock[rdc]==4'b1110)?ALUo_EX:
            (reg_lock[rdc]==4'b1101)?ALUo_MEM:
            (reg_lock[rdc]==4'b1100)?rdd:
            (reg_lock[rdc]==4'b1001)?Data_MEM:
            (reg_lock[rdc]==4'b1000)?rdd:array_reg[rdc];

wire rs_conflict,rt_conflict,rd_conflict;
assign rs_conflict=
(doing_op==`addu||
doing_op==`add||
doing_op==`subu||
doing_op==`sltu||
doing_op==`addi||
doing_op==`addiu||
doing_op==`sw||
doing_op==`bne||
doing_op==`beq||
doing_op==`lw) && (reg_lock[rsc]==4'b1010);
```

```
    assign rt_conflict=
    (doing_op==`addu||
    doing_op==`add||
    doing_op==`subu||
    doing_op==`sltu||
    doing_op==`sw||
    doing_op==`bne||
    doing_op==`beq||
    doing_op==`sll)  && (reg_lock[rtc]==4'b1010);

    assign rd_conflict=(0)&& (reg_lock[rdc]==4'b1010);

    assign detect_conflict=rs_conflict||rt_conflict||rd_conflict;//这里不需要用这个信
endmodule
```

#### 10.1.15 测试平台 (246tb_ex10_tb.v)

#### 10.1.16 自动化测试脚本 (run_cpu_tests.do)

```
# ModelSim Batch Script for CPU Pipeline Testing

# Clean up any existing libraries and files
if {[file exists work]} {
    vdel -lib work -all
}
# Don't delete transcript and vsim.wlf as they might be in use
# Clean up any old result files if they exist
catch {file delete _246tb_ex10_result.txt}

# Create fresh working library
vlib work

# Compile IP core files first (these are required for the design to work)
vlog -work work -quiet ./cpupip8.srcs/sources_1/ip/dmem1/dist_mem_gen_v8_0_10/simulat
vlog -work work -quiet ./cpupip8.srcs/sources_1/ip/dmem1/sim/dmem1.v

# Compile source files
```

```
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/def.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/alu.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/BJudge.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/PCreg.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/DMEM.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/EX_MEM.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/ID_EX.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/IF_ID.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/IMEM.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/MEM_WB.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/NPCmaker.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/regfile.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/cpu.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/sccomp_dataflow.v
vlog -work work -quiet ./cpupip8.srcs/sim_1/new/_246tb_ex10_tb.v

# Create results directory using Tcl commands
set results_dir "./test_scripts/results"
if {![file exists $results_dir]} {
    file mkdir $results_dir
}

# List of all test files to run
set test_files [list \
    "testdata/1_addi.hex.txt" \
    "testdata/2_addiu.hex.txt" \
    "testdata/9_addu.hex.txt" \
    "testdata/11_beq.hex.txt" \
    "testdata/12_bne.hex.txt" \
    "testdata/16.26_lwsw.hex.txt" \
    "testdata/16.26_lwsw2.hex.txt" \
    "testdata/20_sll.hex.txt" \
    "testdata/22_sltu.hex.txt" \
    "testdata/25_subu.hex.txt" \
]

# Procedure to run a single test
```

```
proc run_single_test {test_file results_dir} {
    # Get test name without extension
    set test_name [file rootname [file tail $test_file]]
    set test_name [string map {.hex ""} $test_name]

    puts "\n------------------------------"
    puts "RUNNING TEST: $test_file"
    puts "------------------------------"

    # Clean up any old result files before starting the test
    if {[file exists ./_246tb_ex10_result.txt]} {
        catch {file delete ./_246tb_ex10_result.txt}
    }

    # Backup original IMEM.v
    file copy -force ./cpupip8.srcs/sources_1/new/IMEM.v ./cpupip8.srcs/sources_1/new/

    # Read the original IMEM.v
    set fid [open "./cpupip8.srcs/sources_1/new/IMEM.v" r]
    set content [read $fid]
    close $fid

    # Replace the readmemh line with the current test file
    set lines [split $content "\n"]
    set new_lines {}
    foreach line $lines {
        if {[string match "*\$readmemh*" $line] && [string match "*IMEMreg*" $line] &
            # Found the active $readmemh line, replace it
            lappend new_lines "       \$readmemh(\"E:/Homeworks/cpupip8/$test_file\",
        } else {
            lappend new_lines $line
        }
    }

    set new_content [join $new_lines "\n"]

    # Write the modified IMEM.v
```

```
set fid [open "./cpupip8.srcs/sources_1/new/IMEM.v" w]
puts -nonewline $fid $new_content
close $fid

# Recompile only IMEM.v
vlog -work work -quiet ./cpupip8.srcs/sources_1/new/IMEM.v

# Load and run simulation
vsim -quiet work._246tb_ex10_tb

# Run simulation for maximum 100000ns, but check for halt condition
run 100000ns

# Copy the test result file to the results directory
set sim_result_file "./_246tb_ex10_result.txt"
set output_result_file "$results_dir/${test_name}_sim_result.txt"

if {[file exists $sim_result_file]} {
    file copy -force $sim_result_file $output_result_file
    puts "Saved simulation result to: $output_result_file"
} else {
    puts "Warning: Simulation result file not found: $sim_result_file"
}

# Restore original IMEM.v
file copy -force ./cpupip8.srcs/sources_1/new/IMEM.v.bak ./cpupip8.srcs/sources_1
file delete ./cpupip8.srcs/sources_1/new/IMEM.v.bak

# Generate standard result using external simulator
set cmd_result [catch {exec ./cpu_pipelined_simulator.exe E:/Homeworks/cpupip8/$t
if {$cmd_result == 0} {
    puts "Standard test result generated successfully for $test_file"
} else {
    puts "Failed to generate standard test result for $test_file: $std_output"
    return 0
}
```

```
# Generate standard result using external simulator and save it
set std_result_file "$results_dir/${test_name}_std_result.txt"
set cmd_result [catch {exec ./cpu_pipelined_simulator.exe E:/Homeworks/cpupip8/$t
if {$cmd_result == 0} {
    puts "Standard test result saved to: $std_result_file"
} else {
    # Fallback: run without explicit output file and capture output
    set cmd_result [catch {exec ./cpu_pipelined_simulator.exe E:/Homeworks/cpupip
    if {$cmd_result == 0} {
        set std_fid [open $std_result_file w]
        puts $std_fid $std_output_temp
        close $std_fid
        puts "Standard test result saved to: $std_result_file from captured outpu
    } else {
        puts "Failed to generate standard test result for $test_file: $std_output
        return 0
    }
}

# Compare simulation result with standard result using external tool
set compare_result [catch {exec txt_compare --file1 $output_result_file --file2 $

# Check the comparison result
set comp_file "$results_dir/${test_name}_comparison_result.txt"
if {[file exists $comp_file]} {
    set comp_fid [open $comp_file r]
    set comp_content [read $comp_fid]
    close $comp_fid

    if {[string match "*PASS*" $comp_content] || [string match "*MATCH*" $comp_co
        puts "RESULT: PASS - $test_file"
        return 1
    } else {
        puts "RESULT: FAIL - $test_file"
        return 0
    }
} else {
        puts "Comparison result file not found: $comp_file"
        return 0
    }
}

# Run all tests
set total_tests [llength $test_files]
set pass_count 0

puts "Starting batch simulation for $total_tests tests..."

foreach test_file $test_files {
    set result [run_single_test $test_file $results_dir]
    if {$result == 1} {
        incr pass_count
    }
}

# Generate final summary
set summary_text [subst "BATCH TEST SUMMARY\n==================================\nTota

puts "\n$summary_text"

# Write summary to file
set sum_fid [open "$results_dir/test_summary.txt" w]
puts $sum_fid $summary_text
close $sum_fid

puts "\nAll tests completed. Results saved in $results_dir/"
puts "Success rate: [format %.2f [expr double($pass_count)*100/double($total_tests)]]

# Quit ModelSim
quit
```

### 10.1.17 软件仿真器 (cpu_pipelined_simulator.cpp)

```
#include <iostream>
#include <fstream>
```

```
#include <vector>
#include <iomanip>
#include <sstream>
#include <string>
#include <cstdint>

#define DEBUG_BRANCH 0
#define DEBUG_CONFLICT 1

using namespace std;

class ExactPipelinedCPUSimulator {
private:
    vector<int32_t> registers;
    vector<int32_t> instruction_memory;

    struct PipelineStage {
        bool valid;
        int32_t pc;
        int32_t instr;
        uint32_t dest_reg;
        int32_t result;
        uint32_t rs, rt, rd;
        uint8_t op_type; // 0: 非ALU, 1: ALU操作, 2: 内存加载

        PipelineStage() : valid(false), pc(0), instr(0), dest_reg(0),
                          result(0), rs(0), rt(0), rd(0), op_type(0) {}
    };

    PipelineStage pipe[5];
    int32_t display_pc;
    int32_t actual_pc;

    // 寄存器锁机制
    struct RegisterLock {
        uint8_t lock_type; // 高2位: 0-无锁, 1-ALU型, 2-内存型; 低2位: 计时器
        int32_t value;     // 当前阶段的值
        bool valid;        // 锁是否有效

        RegisterLock() : lock_type(0), value(0), valid(false) {}
    };

    vector<RegisterLock> register_locks;

    // 操作类型定义
    enum OpType {
        OP_ALU = 1,
        OP_MEM = 2
    };

    void decode_instruction(int32_t instr, uint8_t& opcode, uint8_t& funct,
                          uint32_t& rs, uint32_t& rt, uint32_t& rd, uint32_t& shamt,
        opcode = (instr >> 26) & 0x3F;
        rs = (instr >> 21) & 0x1F;
        rt = (instr >> 16) & 0x1F;
        rd = (instr >> 11) & 0x1F;
        shamt = (instr >> 6) & 0x1F;
        imm = instr & 0xFFFF;
        funct = instr & 0x3F;
    }

    int32_t sign_extend_16_to_32(uint16_t value) {
        if (value & 0x8000) {
            return static_cast<int32_t>(0xFFFF0000 | value);
        } else {
            return static_cast<int32_t>(value);
        }
    }

    int32_t alu_operation(uint8_t alu_op, int32_t a, int32_t b) {
        switch(alu_op) {
            case 0:  return static_cast<uint32_t>(a) + static_cast<uint32_t>(b);  //
            case 1:  return a - b;                                                 //
            case 2:  return a + b;                                                 //
            case 3:  return a - b;                                                 //
            case 4:  return a & b;                                                 //
            case 5:  return a | b;                                                 //
            case 6:  return a ^ b;                                                 //
            case 7:  return ~(a | b);                                              //
            case 8:  return static_cast<int32_t>((b & 0xFFFF) << 16);             //
            case 10: return (static_cast<uint32_t>(a) < static_cast<uint32_t>(b)) ? 1
            case 11: return (a < b) ? 1 : 0;                                      //
            case 12: { // SRA - 算术右移
                int shift = a & 0x1F;
                if (shift >= 32) return (b & 0x80000000) ? 0xFFFFFFFF : 0x00000000;
                else return (b >> shift) | ((b & 0x80000000) ? ((1 << (32-shift)) - 1
            }
            case 13: return static_cast<uint32_t>(b) >> (a & 0x1F);               //
            case 14: return b << (a & 0x1F);                                      //
            default: return 0;
        }
    }

    uint8_t get_alu_op(int32_t instr) {
        uint8_t opcode, funct;
        uint32_t rs, rt, rd, shamt;
        uint32_t imm;
        decode_instruction(instr, opcode, funct, rs, rt, rd, shamt, imm);

        if (opcode == 0x00) { // R-type
            switch(funct) {
                case 0x20: return 2;  // ADD
                case 0x21: return 0;  // ADDU
                case 0x22: return 3;  // SUB
                case 0x23: return 1;  // SUBU
                case 0x24: return 4;  // AND
                case 0x25: return 5;  // OR
                case 0x26: return 6;  // XOR
                case 0x27: return 7;  // NOR
                case 0x2A: return 11; // SLT
                case 0x2B: return 10; // SLTU
                case 0x00: return 14; // SLL
                case 0x02: return 13; // SRL
                case 0x03: return 12; // SRA
                default: return 0;
            }
        } else if (opcode == 0x08) { // ADDI
            return 2;
        } else if (opcode == 0x09) { // ADDIU
            return 0;
        } else if (opcode == 0x0C || opcode == 0x0D || opcode == 0x0E) { // ANDI, ORI
            return (opcode == 0x0C) ? 4 : (opcode == 0x0D) ? 5 : 6;
        } else if (opcode == 0x0F) { // LUI
            return 8;
        } else if (opcode == 0x23 || opcode == 0x2B) { // LW, SW
            return 0;
        } else {
            return 0;
        }
    }

    uint8_t get_op_type(int32_t instr) {
        uint8_t opcode, funct;
        uint32_t rs, rt, rd, shamt;
        uint32_t imm;
        decode_instruction(instr, opcode, funct, rs, rt, rd, shamt, imm);

        if (opcode == 0x00) { // R-type
            return OP_ALU;
        } else if (opcode == 0x08 || opcode == 0x09 || opcode == 0x0C ||
                  opcode == 0x0D || opcode == 0x0E || opcode == 0x0F) { // 立即数ALU
            return OP_ALU;
        } else if (opcode == 0x23) { // LW
            return OP_MEM;
        } else {
            return 0;
        }
    }
```

```
// 获取指令的目标寄存器
uint32_t get_dest_reg(int32_t instr) {
    uint8_t opcode, funct;
    uint32_t rs, rt, rd, shamt;
    uint32_t imm;
    decode_instruction(instr, opcode, funct, rs, rt, rd, shamt, imm);

    if (opcode == 0x00) { // R-type
        return rd;
    } else if (opcode == 0x08 || opcode == 0x09 || opcode == 0x0C ||
              opcode == 0x0D || opcode == 0x0E || opcode == 0x0F ||
              opcode == 0x23) { // 立即数ALU指令或LW
        return rt;
    } else {
        return 0;
    }
}

// 更新寄存器锁
void update_register_locks() {
    // 首先减少所有锁的计时器
    for (int i = 0; i < 32; i++) {
        if (register_locks[i].valid && register_locks[i].lock_type > 0) {
            uint8_t timer = register_locks[i].lock_type & 0x03;
            if (timer > 0) {
                timer--;
                register_locks[i].lock_type = (register_locks[i].lock_type & 0xFC
            } else {
                // 计时器归零，释放锁
                register_locks[i].valid = false;
                register_locks[i].lock_type = 0;
            }
        }
    }

    // 在EX阶段设置新的锁
    if (pipe[2].valid && pipe[2].dest_reg != 0) {
        uint32_t reg = pipe[2].dest_reg;
        register_locks[reg].valid = true;
        register_locks[reg].lock_type = (pipe[2].op_type << 2) | 0x03; // 设置类型
        register_locks[reg].value = pipe[2].result;

        #if DEBUG_CONFLICT
        cout << "DEBUG: Setting lock for reg " << dec << reg
             << " type: " << (int)(pipe[2].op_type) << " value: " << hex << pipe[
        #endif
    }

    // 更新MEM和WB阶段的值
    if (pipe[3].valid && pipe[3].dest_reg != 0) {
        uint32_t reg = pipe[3].dest_reg;
        if (register_locks[reg].valid) {
            register_locks[reg].value = pipe[3].result;
        }
    }

    if (pipe[4].valid && pipe[4].dest_reg != 0) {
        uint32_t reg = pipe[4].dest_reg;
        if (register_locks[reg].valid) {
            register_locks[reg].value = pipe[4].result;
        }
    }
}

// 检测数据冲突
bool detect_data_conflict() {
    if (!pipe[1].valid) return false;

    uint8_t opcode, funct;
    uint32_t rs, rt, rd, shamt;
    uint32_t imm;
    decode_instruction(pipe[1].instr, opcode, funct, rs, rt, rd, shamt, imm);
```

```
bool conflict = false;

// 检查rs冲突
if (rs != 0 && register_locks[rs].valid) {
    uint8_t lock_type = register_locks[rs].lock_type;
    uint8_t op_type = lock_type >> 2;
    uint8_t timer = lock_type & 0x03;

    // 如果是内存操作且计时器为2（在EX阶段），则发生冲突
    if (op_type == OP_MEM && timer == 2) {
        #if DEBUG_CONFLICT
        cout << "DEBUG: RS conflict detected for reg " << dec << rs
             << " lock_type: " << (int)lock_type << endl;
        #endif
        conflict = true;
    }
}

// 检查rt冲突（对于需要rt的指令）
if ((opcode == 0x00 || opcode == 0x04 || opcode == 0x05 ||
     opcode == 0x08 || opcode == 0x09 || opcode == 0x23 ||
     opcode == 0x2B) && rt != 0 && register_locks[rt].valid) {
    uint8_t lock_type = register_locks[rt].lock_type;
    uint8_t op_type = lock_type >> 2;
    uint8_t timer = lock_type & 0x03;

    // 如果是内存操作且计时器为2（在EX阶段），则发生冲突
    if (op_type == OP_MEM && timer == 2) {
        #if DEBUG_CONFLICT
        cout << "DEBUG: RT conflict detected for reg " << dec << rt
             << " lock_type: " << (int)lock_type << endl;
        #endif
        conflict = true;
    }
}

return conflict;
```

```
    }

    // 通过重定向获取寄存器值
    int32_t get_register_value_with_forwarding(uint32_t reg) {
        if (reg == 0) return 0;

        if (!register_locks[reg].valid) {
            return registers[reg];
        }

        uint8_t lock_type = register_locks[reg].lock_type;
        uint8_t op_type = lock_type >> 2;
        uint8_t timer = lock_type & 0x03;

        // 根据锁的状态进行前递
        switch (timer) {
            case 3: // EX阶段
                return register_locks[reg].value;
            case 2: // MEM阶段
                return register_locks[reg].value;
            case 1: // WB阶段
                return register_locks[reg].value;
            default:
                return registers[reg];
        }
    }

public:
    bool halted;
    bool branch_detected_in_id;
    int32_t branch_target;
    int32_t branch_pc;
    bool insert_bubble_next_cycle;
    bool branch_taken;
    bool is_bubble_cycle;
    bool data_conflict_detected;
```

```
ExactPipelinedCPUSimulator() : registers(32, 0), register_locks(32),
                               halted(false), branch_detected_in_id(false),
                               branch_target(0), branch_pc(0),
                               insert_bubble_next_cycle(false),
                               branch_taken(false), is_bubble_cycle(false),
                               data_conflict_detected(false),
                               display_pc(0), actual_pc(0) {
    registers[0] = 0;
    for (int i = 0; i < 5; i++) {
        pipe[i] = PipelineStage();
    }
}

void load_hex_file(const string& filename) {
    ifstream file(filename);
    string line;

    while (getline(file, line)) {
        if (line.empty()) continue;

        uint32_t hex_val;
        stringstream ss;
        ss << std::hex << line;
        ss >> hex_val;

        instruction_memory.push_back(static_cast<int32_t>(hex_val));
    }
}

void print_state(ostream& out, int32_t current_display_pc) {
    out << "pc: " << setw(8) << setfill('0') << hex << current_display_pc << endl

    if (is_bubble_cycle) {
        out << "instr: " << setw(8) << setfill('0') << hex << 0 << endl;
    } else {
        int addr = current_display_pc / 4;
        if (addr < instruction_memory.size()) {
            out << "instr: " << setw(8) << setfill('0') << hex << instruction_mem
        } else {
            out << "instr: " << setw(8) << setfill('0') << hex << 0 << endl;
        }
    }

    for (int i = 0; i < 32; i++) {
        out << "regfile" << dec << i << ": " << setw(8) << setfill('0') << hex <<
    }
}

void execute_cycle() {
    if (halted) {
        return;
    }

    // 重置标志
    is_bubble_cycle = false;
    data_conflict_detected = false;

    // 1. 更新寄存器锁
    update_register_locks();

    // 2. 检测数据冲突
    data_conflict_detected = detect_data_conflict();

    if (data_conflict_detected) {
        #if DEBUG_CONFLICT
        cout << "DEBUG: Data conflict detected, stalling pipeline" << endl;
        #endif

        // 插入气泡，停止PC
        pipe[0].valid = false;
        insert_bubble_next_cycle = true;
        is_bubble_cycle = true;

        // 保持显示PC不变
        display_pc = actual_pc;

        return;
    }

    // 3. WB阶段：更新寄存器
    if (pipe[4].valid && pipe[4].dest_reg != 0) {
        registers[pipe[4].dest_reg] = pipe[4].result;
    }

    // 4. 移动流水线
    for (int i = 4; i > 0; i--) {
        pipe[i] = pipe[i-1];
    }

    // 5. ID阶段：分支检测和指令解码
    if (pipe[1].valid) {
        uint8_t opcode, funct;
        uint32_t rs, rt, rd, shamt;
        uint32_t imm;
        decode_instruction(pipe[1].instr, opcode, funct, rs, rt, rd, shamt, imm);

        #if DEBUG_BRANCH
        cout << "DEBUG: ID stage - PC: " << hex << pipe[1].pc
             << " Instr: " << pipe[1].instr << " Opcode: " << (int)opcode << endl
        #endif

        // 检查HALT指令
        if (pipe[1].instr == 0xffffffff) {
            halted = true;
        }
        // 在ID阶段处理分支指令
        else if (opcode == 0x04 || opcode == 0x05) { // BEQ 或 BNE
            // 使用重定向后的寄存器值
            int32_t a = get_register_value_with_forwarding(rs);
            int32_t b = get_register_value_with_forwarding(rt);
            int32_t offset = sign_extend_16_to_32(imm) << 2;
            #if DEBUG_BRANCH
            cout << "DEBUG: Branch detected in ID, a=" << a << " b=" << b << " of
                 << " branch_pc=" << hex << pipe[1].pc << " target=" << (pipe[1].
            #endif

            branch_target = pipe[1].pc + 4 + offset;
            branch_pc = pipe[1].pc;
            branch_detected_in_id = true;
            insert_bubble_next_cycle = true;

            if ((opcode == 0x04 && a == b) || (opcode == 0x05 && a != b)) {
                branch_taken = true;
                #if DEBUG_BRANCH
                cout << "DEBUG: Branch taken! Will update actual_pc to " << hex <
                #endif
            } else {
                branch_taken = false;
                #if DEBUG_BRANCH
                cout << "DEBUG: Branch not taken!" << endl;
                #endif
            }
        }
    }

    // 6. EX阶段：执行ALU操作
    if (pipe[2].valid) {
        uint8_t opcode, funct;
        uint32_t rs, rt, rd, shamt;
        uint32_t imm;
        decode_instruction(pipe[2].instr, opcode, funct, rs, rt, rd, shamt, imm);

        // 使用重定向后的寄存器值
        int32_t a = get_register_value_with_forwarding(pipe[2].rs);
        int32_t b = get_register_value_with_forwarding(pipe[2].rt);

        // 对于立即数指令，使用符号扩展的立即数作为b
        if (opcode == 0x08 || opcode == 0x09 || opcode == 0x23 || opcode == 0x2B
            opcode == 0x0C || opcode == 0x0D || opcode == 0x0E) {
            b = sign_extend_16_to_32(imm);
        } else if (opcode == 0x0F) { // LUI
            b = static_cast<int32_t>(imm << 16);
        } else if (opcode == 0x00 && funct == 0x00) { // SLL指令
            a = shamt;
        }

        // 执行ALU操作
        if (!halted) {
            uint8_t alu_op = get_alu_op(pipe[2].instr);
            pipe[2].result = alu_operation(alu_op, a, b);
        }
    }

    // 7. 新的IF阶段
    if (insert_bubble_next_cycle) {
        // 插入气泡周期
        pipe[0].valid = true;
        pipe[0].pc = branch_pc;
        pipe[0].instr = 0x00000000;
        pipe[0].dest_reg = 0;
        pipe[0].rs = 0;
        pipe[0].rt = 0;
        pipe[0].rd = 0;
        pipe[0].op_type = 0;

        display_pc = branch_pc;
        insert_bubble_next_cycle = false;
        is_bubble_cycle = true;

        if (branch_taken) {
            actual_pc = branch_target;
        } else {
            actual_pc = branch_pc + 4;
        }
```

```
} else {
    // 正常取指
    int next_instr_addr = actual_pc / 4;
    if (next_instr_addr < instruction_memory.size() && !halted) {
        int32_t next_instr = instruction_memory[next_instr_addr];

        uint8_t opcode, funct;
        uint32_t rs, rt, rd, shamt;
        uint32_t imm;
        decode_instruction(next_instr, opcode, funct, rs, rt, rd, shamt, imm)

        // 检查分支指令
        if (opcode == 0x04 || opcode == 0x05) {
            int32_t offset = sign_extend_16_to_32(imm) << 2;
            branch_target = actual_pc + 4 + offset;
            branch_pc = actual_pc;
            branch_detected_in_id = true;
            insert_bubble_next_cycle = true;

            int32_t a = get_register_value_with_forwarding(rs);
            int32_t b = get_register_value_with_forwarding(rt);
            if ((opcode == 0x04 && a == b) || (opcode == 0x05 && a != b)) {
                branch_taken = true;
            } else {
                branch_taken = false;
            }
        }

        if (next_instr == 0xffffffff) {
            pipe[0].valid = true;
            pipe[0].pc = actual_pc;
            pipe[0].instr = next_instr;
            pipe[0].rs = rs;
            pipe[0].rt = rt;
            pipe[0].rd = rd;
            pipe[0].dest_reg = 0;
            pipe[0].op_type = 0;
```

```
                halted = true;
            } else {
                pipe[0].valid = true;
                pipe[0].pc = actual_pc;
                pipe[0].instr = next_instr;
                pipe[0].rs = rs;
                pipe[0].rt = rt;
                pipe[0].rd = rd;

                uint32_t dest_reg = get_dest_reg(next_instr);
                pipe[0].dest_reg = dest_reg;
                pipe[0].op_type = get_op_type(next_instr);
            }

            display_pc = actual_pc;
            actual_pc += 4;
        } else {
            pipe[0].valid = false;
            pipe[0].pc = actual_pc;
            pipe[0].instr = 0;
            pipe[0].dest_reg = 0;
            pipe[0].op_type = 0;

            display_pc = actual_pc;
            actual_pc += 4;
        }
    }

    // 确保$0寄存器始终为0
    registers[0] = 0;
}

void simulate(const string& input_file, const string& output_file) {
    load_hex_file(input_file);

    ofstream outfile(output_file);
```

```
        int total_cycles = instruction_memory.size() + 10;

        for (int cycle = 0; cycle < total_cycles; cycle++) {
            execute_cycle();
            print_state(outfile, display_pc);

            if (halted) {
                break;
            }
        }
    }
};

int main(int argc, char* argv[]) {
    ExactPipelinedCPUSimulator cpu;

    string input_file = "test_custom_instructions.hex.txt";
    string output_file = "test_output.result.txt";

    if (argc >= 2) {
        input_file = argv[1];
    }
    if (argc >= 3) {
        output_file = argv[2];
    }

    cpu.simulate(input_file, output_file);

    cout << "精确流水线CPU模拟完成。结果已保存到 " << output_file << endl;
    return 0;
}
```