山东大学 计算机科学与技术 学院

计算机体系结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： | 姓名： | | 班级： |
| 实验题目：实验六 指令调度 | | | |
| 实验学时：4 | | 实验日期： 2025.5.28 | |
| 实验目的：  通过本实验，加深对指令调度的理解，了解指令调度技术对 CPU 性能改进的好处。 | | | |
| 硬件环境：  CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz 2.81 GHz  RAM: 24.0 GB | | | |
| 软件环境：  Windows 11 专业版  Ovtdm 0.9.0  WinDLX | | | |
| 实验步骤与内容：  （1） 通过 Configuration 菜单中的“Floating point stages”选项，把除法单元数设置为3，把加法﹑乘法﹑除法的延迟设置为 3 个时钟周期。  （2） 用 WinDLX 模拟器运行调度前的程序 sch-before.s 。记录程序执行过程中各种相关发生的次数以及程序执行的总时钟周期数。  （3） 用 WinDLX 模拟器运行调度后的程序 sch-after.s ，记录程序执行过程中各种相关发生的次数以及程序执行的总时钟周期数。  （4） 根据记录结果，比较调度前和调度后的性能。  （5） 论述指令调度对于提高 CPU 性能的意义。  首先关闭设置 FPS 选项，运行调度前的程序：    调度前的代码中，连续的divf指令（如divf f1,f8,f7→divf f2,f9,f7→divf f10,f3,f7等）因资源争用和数据依赖链导致流水线频繁停顿：   * 资源争用：浮点除法单元延迟为3周期，但连续6个divf指令串行执行，无法并行，导致每个divf需等待前一个完成，形成结构性停顿。 * 数据依赖链：addf f3,f1,f2需等待前两个divf完成，后续divf f10,f3,f7又依赖addf的结果，形成长依赖链，引发大量RAW stalls。     调度后的代码通过指令重排优化了这两个问题：  打破依赖链：将addf f3,f1,f2和multf f6,f4,f5移至所有divf之后，使其无需等待中间结果，减少RAW stalls。  并行执行独立指令：将divf f4,f11,f7和divf f5,f12,f7提前，与前两个divf并行执行，充分利用3个浮点除法单元的并行能力（如第1-3周期执行divf f1、divf f2、divf f4，第4-6周期执行divf f5、divf f10、divf f13等），减少资源争用。  代码分析：  ;-------------------------------------------------------------------  ; 调度前的程序 sch-before.s：连续浮点除法（divf）指令因资源争用和数据依赖导致流水线停顿  ;-------------------------------------------------------------------  lf f1, ONE ; 加载常量1到f1  cvti2f f7, f1 ; 将f1的整数1转换为浮点格式存入f7（作为除法分母）  nop ; 空操作，用于对齐指令  divf f1, f8, f7 ; f1 = f8 / f7（Y = f8 / 1 → Y = f8）  divf f2, f9, f7 ; f2 = f9 / f7（Z = f9 / 1 → Z = f9）  addf f3, f1, f2 ; f3 = f1 + f2（X = Y + Z）  divf f10, f3, f7 ; f10 = f3 / f7（X = (Y+Z)/1 → X = Y+Z）  divf f4, f11, f7 ; f4 = f11 / f7（B = f11 / 1 → B = f11）  divf f5, f12, f7 ; f5 = f12 / f7（C = f12 / 1 → C = f12）  multf f6, f4, f5 ; f6 = f4 \* f5（A = B \* C）  divf f13, f6, f7 ; f13 = f6 / f7（A = (B\*C)/1 → A = B\*C）  ;-------------------------------------------------------------------  ; 调度后的程序 sch-after.s：通过指令调度减少数据相关和资源冲突  ;-------------------------------------------------------------------  lf f1, ONE ; 加载常量1到f1  cvti2f f7, f1 ; 将f1的整数1转换为浮点格式存入f7  nop ; 空操作  divf f1, f8, f7 ; f1 = f8 / f7（Y = f8）  divf f2, f9, f7 ; f2 = f9 / f7（Z = f9）  divf f4, f11, f7 ; f4 = f11 / f7（B = f11）  divf f5, f12, f7 ; f5 = f12 / f7（C = f12）  addf f3, f1, f2 ; f3 = f1 + f2（X = Y+Z）  multf f6, f4, f5 ; f6 = f4 \* f5（A = B\*C）  divf f10, f3, f7 ; f10 = f3 / f7（X = Y+Z）  divf f13, f6, f7 ; f13 = f6 / f7（A = B\*C） | | | |
| 结论分析与体会：  通过WinDLX模拟器运行调度前后的程序，验证了指令调度对CPU性能的显著提升效果。调度前的代码中，连续divf指令因资源争用（如浮点除法单元需3周期）和长数据依赖链（如addf依赖前序divf结果）导致流水线频繁停顿（RAW stalls占51.35%），总周期数达37。调度后通过重排独立指令（如将divf f4,f11,f7与divf f5,f12,f7提前），打破依赖链并充分利用3个浮点除法单元的并行能力，使RAW stalls减少至33.33%，总周期数降至27，性能提升约1.37倍。这一过程直观体现了指令调度在减少资源冲突、提升硬件利用率中的关键作用，也表明合理安排指令顺序是优化浮点密集型计算效率的核心手段。 | | | |