《计算机组成与设计》课程总结

李清安

第一章 计算机抽象与技术

- 1.1 引言——计算机的分类
- 1.2 **计算机结构中的7个伟大思路**
- 1.3 软件到硬件的流程
  - **编译、汇编、链接、加载（参考2.12节）**
- 1.4 箱盖下的硬件
- **1.6 性能**
  - CPU性能的定义
  - **延时 与 吞吐量（带宽）**
  - **性能 与 执行时间 与 CPI**
  - T = #cylces * cycle-time = #insts * CPI * cycle-time
    - 例子、习题
    - 考试之外，有何意义——联系算法课上的性能
- 1.7-1.8 功耗墙 与多处理器
- 1.9 CPU性能的评测
- 1.10 矩阵乘法的性能优化

第2章 指令——计算机的语言

- 2.2 **计算机硬件的操作码**
  - 图2.1 常用指令：算术逻辑运算指令、内存访问指令、跳转指令
- 2.3 **计算机硬件的操作数**
  - 寄存器、内存地址、常量
  - 例子（与C语言对应）
- 2.4 **整数的表示**
  - 2进制、10进制、16进制，进制转换
  - 无符号、补码
- 2.5 **指令的表示**
  - 算术指令与C语言的对应
  - R指令、I指令、S指令的格式与特点（图2-5）
- 2.6 **逻辑指令**
  - 逻辑指令与C语言的对应（比如整数乘法运算用移位指令实现）
    - 注意带符号右移

第2章 指令——计算机的语言

- 2.7 **分支指令**
  - beq、bne
  - slt、slti （图2.41）
  - **分支指令与C语言中if-else、循环的对应**
- 2.8 **过程调用指令**
  - jal ——call、jalr——return
  - 寄存器约定：参数a0-a7、返回值a0-a1、返回地址ra
  - **过程调用的过程、例子**
    - 寄存器保护（saving寄存器）
    - 栈空间分配 （比如addi sp, sp, -8）
    - 读取参数、工作、写返回值
    - 栈空间释放 (比如addi sp, sp, 8)
    - 寄存器恢复
    - jalr ra
  - **递归函数（p74）**

Caller和Callee的分工——完整版

caller	callee
• ...准备调用**callee** 1. **调用前**，保存caller-saving寄存器到caller栈上（t0～t6, ra, a0~a7) 2. 将callee所需实际参数准备好，即拷贝到约定的参数位置（a0~a7） 3. **调用时**，jal x1， calleeEntry指令：将返回地址填入约定的x1寄存器，同时将PC寄存器更新为calleeEntry 4. **调用后**，从callee约定的返回值寄存器（a0）获取返回值 5. 从栈上恢复caller-saving寄存器 • 往后执行...	a) 分配栈空间 b) 保护callee-saving寄存器到栈上(s0~s11) c) 从约定的位置（a0～a7）读取参数值 d) 完成计算任务 e) 将结果存储到约定的返回值寄存器（a0） f) 从栈上恢复callee-saving寄存器 g) 释放栈空间 h) 利用jalr x0， 0(x1)指令，跳转到约定的返回地址寄存器（x1）

第2章 指令——计算机的语言

- 2.9 字符串的表示与运算
- 2.10 **常量与寻址模式**
  - **beq/bne/jal指令中的常量偏移需要*2**
    - 例子
  - 大常量的加载需多条指令
  - **RISC-V支持的寻址模式**
    - **图2.17，图2.18**
- 2.11 并行与同步指令（lr.w, sc.w）
- 2.12 **程序的翻译与运行流程**
  - 编译、汇编、链接、加载（图2.20）
- 2.13 **排序的例子——根据汇编猜出对应的C代码?**
- 2.14 **数组与指针**

第3章 计算机的算术运算

- 3.2 整数的加法和减法
  - 减法通过加法实现
  - 加法、减法运算的溢出检测（图3-2）
- 3.3 乘法
- 3.5 **浮点表示及浮点加法的运算过程**
  - **IEEE浮点格式（图3.13）**
    - **规格化数、非规格化数、NaN**
    - **如何确定浮点格式的表示范围**
    - **上溢（overflow、指数部分）、下溢（underflow、小数部分）**
  - **浮点加法过程——对齐、相加、规格化、舍入）**
    - **例子、图3.14**
  - 3.5 浮点乘法过程
- 3.6 向量指令与字内并行
- 3.8 矩阵乘法的性能优化

第4章 处理器

- 4.1-4.2：**CPU的基本组件及其特点**
  - 与门、或门、非门的常用符号
  - 理解时钟、状态逻辑与组合逻辑
- 4.3 单周期CPU的数据通路
  - 通路：Register、Memory、选择器、**ALU**、Adder
  - 控制：**ALU**、Registers、Memory、选择器（0/1的含义）
  - 基本的CPU通路（图4.11）
- 4.4：**单周期CPU的实现——加上控制**
  - ALU控制的实现（图4.19）
  - control信号的理解：图4.20
  - 主控制的实现（图4.21）
  - **典型指令的datapath、control取值：图4.23-4.25**
  - **如果设计一条新指令，是否需要增加组件、修改控制信号?**
  - 单周期CPU的CPI理解

第4章 处理器

- 4.6 **流水线CPU概述**
  - 五阶段：IF、ID、EXE、MEM、WB
  - 流水线CPU的加速比、CPI，跟指令系统的关系(p198)
  - 流水支持多条指令并行→但是受依赖或冒险影响
  - 结构冒险(p199)
    - 指令间共用资源-> 多份资源、停顿
  - 数据冒险(p200)
    - 指令间数据依赖->旁路(图4.31）、reorder、停顿（图4.32）
  - 控制冒险(p202)
    - 指令间控制依赖->分支结果提前、旁路（图4.33）、reorder、停顿
    - 分支预测（图4.34）

第4章 处理器

- 4.7 **流水线CPU的实现**
  - datapath：
    - 原有datapath + pipeline registers， 图4.37，图4.38-4.42
    - 图4.43，流水线通路的修正
  - control：
    - 原有control+pipeline逐级传递，图4.52，图4.53
- 4.8 **数据冒险的改进——旁路与停顿的实现**
  - R-R数据冒险（图4.54）——旁路（图4.55）
  - 两种旁路的实现：**在EX阶段处理**，图4.56-图4.58
    - EX旁路：从EX/MEM取数据
    - MEM旁路：从MEM/WB取数据
    - 有/无旁路情况下，指令之间需要多少停顿?
  - load-use数据冒险——停顿，**在ID阶段处理**
    - 冒险检测单元及停顿的实现，图4.60-图4.61
    - repeat PC & IF/ID + clear ID/EX

第4章 处理器

- 4.9 **控制冒险**
  - 分支结果提前——减少分支停顿
    - 控制冒险的停顿的实现（4.9.2节）
    - 图4.63-图4-64
  - 分支预测
    - 静态预测——假设分支不转移
    - 1-bit和2-bit动态预测器
    - **给定分支执行序列，计算给定预测器策略上的hit和miss?**
    - **预测失败，则停顿（ID阶段检测）**
      - clear 进入流水的后续指令；refetch正确的跳转目标指令
- 4.10 异常
- 4.11 指令级并行——多发射
- 4.13 矩阵乘法的性能优化

第5章 存储层次

- 5.1 **局部性与存储层次、hit和miss**
- 5.2 存储技术（SRAM、DRAM、Flash、disk）
  - Disk访问时间的计算
    - 寻道时间+找扇区的时间+传输时间
- **5.3 cache基础**
  - cache的结构、缓存容量、块大小、tag、index、valid、data
  - cache的直接映射
    - **块地址% 缓存块数量 （图5.9-5.10）**
    - 字节地址与块地址的转换（p284的例子）
  - 给定一个字节地址序列或者块地址序列，给定cache配置，计算hit ratio，分析地址的组成（哪几位是index？）
    - **连续写**能否命中cache，考虑写操作大小与cache block大小的比值
  - 缓存写
    - 写策略：写回、写直达（直写）
    - 写分配：write-allocate， no write-allocate
  - cache miss的处理

第5章 存储层次

- **5.4 cache的性能与优化**
  - **平均访存时间 =命中时间 + 失效率 * 失效惩罚**
  - CPI 的新公式（考虑访问性能）
    - = CPI-base + memory-ratio * 失效率 * 失效惩罚
    - = CPI-base + memory-ratio * （**平均访存时间** - hit latency or 1)?
  - 降低失效率——组相联
    - 映射机制：缓存块地址% 缓存组的数量（注意区分k-路缓存）
    - cache miss的处理：替换算法
    - 增加块大小
  - 降低失效惩罚——多级缓存
  - 降低命中时间——小的L1缓存
  - 给定一个字节地址序列或者block序列，给定cache配置和替换算法，计算命中率
  - 软件方法

第5章 存储层次

- 5.5 可靠的存储层次
  - 海明距离、偶校验、SEC/DED
- 5.7 **虚拟内存**
  - 虚拟地址到物理地址的翻译（图5.24）
    - 页表的结构（图5.25、图5.26）
    - 页缺失的处理：替换算法LRU（图5.27）
    - 页表的存储优化——多级页表（图5.28）
  - 物理地址翻译的加速——页表的缓存：TLB（图5.29）
  - 虚拟地址—物理地址—Cache（TLB+页表+Cache+内存）
    - 虚拟地址->TLB/页表，得到物理地址；物理地址-> Cache/内存
    - TLB、页表、Cache的hit/miss（图5.32、第5.7.9节）
- 5.8 Cache和虚拟内存的统一描述
  - 二者在替换策略、写策略、映射策略等的不同选择（第5.8.1-5.8.4节）
  - 三种失效分类（第5.8.5节）
- 5.15 矩阵乘法的性能优化