前瞻——课程思路、备考技巧与性能衡量

课程介绍

你将用数逻中学会的元件，结合你即将学习的知识，设计出一台简单的计算核心。

1. 我们需要做的显然是将需要计算的信息抽象为数据和运算步骤，这便是数据和指令。
2. 有了数据和指令，我们自然需要设计一套通用的计算部件来执行指令计算数据。
3. 这套部件不够快，怎么优化？
4. 数据和指令不是凭空存在的，他们在哪里？
5. 真实的世界和我们的抽象有什么不同？
6. 最后，如何衡量我们的系统好不好呢？

我们需要的	我们将学习的	知识点	章节
我们可以计算什么？	数据以什么形式存在	各种编码方式	chapter3
我们要实现什么计算功能？	汇编指令	RISC-V 指令集	chapter2
如何将计算步骤抽象为功能？	人肉汇编	将C语言转换为指令	chapter2
如何进行数据的简单计算？	设计计算单元	加法、减法、乘法计算单元	chapter3
如何执行一切指令？	设计CPU	单周期CPU	chapter4
如何加快指令的计算？	并行计算指令	流水线 CPU	chapter4

我们需要的	我们将学习的	知识点	章节
快！更快！	优化并行计算的问题	指令前递与分支预测	chapter4
怎样储存数据?	内存与cache	cache的多级结构与计算	chapter5
与真实的世界交互！		IO与RAID	
如何衡量我们的工作?	衡量标准	CPI与8条准则	chapter1

备考策略

1. 实践中记忆，理解性记忆。
2. cheating sheet 没用但是不能没有！复习、安心加查表
3. 历年卷和书后习题十分有用

面向对象

性能衡量

定义

总运行时间: CPU Time + Elapsed time = Execution Time,当然我们做题时绝大多数时候都是只计算 CPU Time

那么 CPU 的时间显然等于总周期乘单周期时间

$$CPU\ Time = CPU\ Clock\ Cycles \times Clock\ Cycle\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

CPU 周期数可以用 CPI 表示

$$\begin{aligned} Clock\ Cycles &= Instruction\ Count \times Cycles\ per\ Instrction \\ CPU\ Time &= Instruction\ Count \times CPI \times Clock\ Cycle\ Time \\ &= \frac{Instruction\ Count\ CPI}{Clock\ Rate} \end{aligned}$$

吞吐量：throughout， 表示一秒内完成的指令数量

例题1：

Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D).

- P1 with a clock rate of 4 GHz and CPIs of 3, 5, 4, and 4,
- P2 with a clock rate of 2.5 GHz and CPIs of 1, 3, 3, and 3.

Given a program with a dynamic instruction count of 2.0×10^6 instructions divided into classes as follows:

- 10% class A,
- 20% class B,
- 40% class C,
- 30% class D.

a. What is the global CPI for each implementation?

b. Find the execution time required in both cases.

唯一衡量标准:

$$Performance = \frac{1}{Execution\ Time}$$

Amdahl's law

显然有些顽固分子无法加速,怎么办呢

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

八条准则

1. Moore'Law：The integrate circuit resource double every 18-24 months.
2. Use Abstraction to Simplify Design
3. Make the common case fast
4. Performance via Parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierarchy of Memory
8. Dependability via Redundancy