计算机体系结构

张燕咏

yanyongz@ustc.edu.cn

Welcome to Computer Architecture

- **主讲：张燕咏 (yanyongz@ustc.edu.cn) 科技楼106**

- **辅讲：闫宇博（yuboyan@ustc.edu.cn) 图书馆1408**

- **课程主页: http://staff.ustc.edu.cn/~comparch**

助教

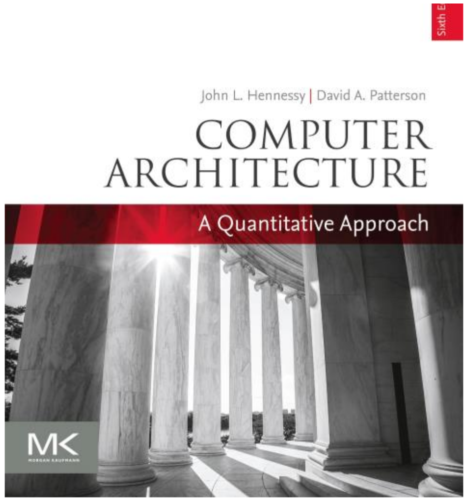
姓名	电子邮件
祝含颀	zhuhanqi@mail.ustc.edu.cn
尤国良	glyou@mail.ustc.edu.cn
翟轶	zhaiyi0@mail.ustc.edu.cn

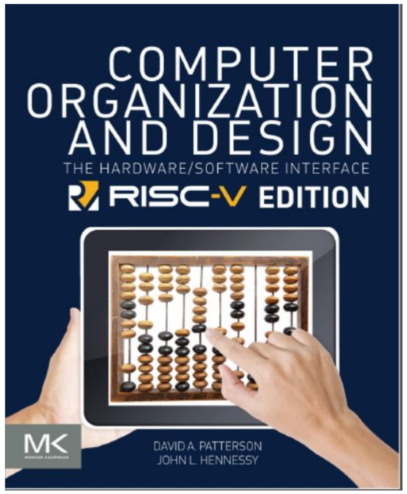
教材与主要参考书

John Leroy Hennessy (born September 22, 1952) is an American computer scientist, academician, businessman, and Chair of Alphabet Inc.[5] Hennessy is one of the founders of MIPS Computer Systems Inc. as well as Atheros and served as the tenth President of Stanford University. Hennessy announced that he would step down in the summer of 2016. He was succeeded as President by Marc Tessier-Lavigne.[6] Marc Andreessen called him "the godfather of Silicon Valley."[7]

Along with David Patterson, Hennessy won the 2017 Turing Award for their work in developing the reduced instruction set computer (RISC) architecture, which is now used in 99% of new computer chips.[8]

David Andrew Patterson (born November 16, 1947) is an American computer pioneer and academic who has held the position of Professor of Computer Science at the University of California, Berkeley since 1976. He announced retirement in 2016 after serving nearly forty years, becoming a distinguished engineer at Google.[3][4] He currently is Vice Chair of the Board of Directors of the RISC-V Foundation,[5] and the Pardee Professor of Computer Science, Emeritus at UC Berkeley.

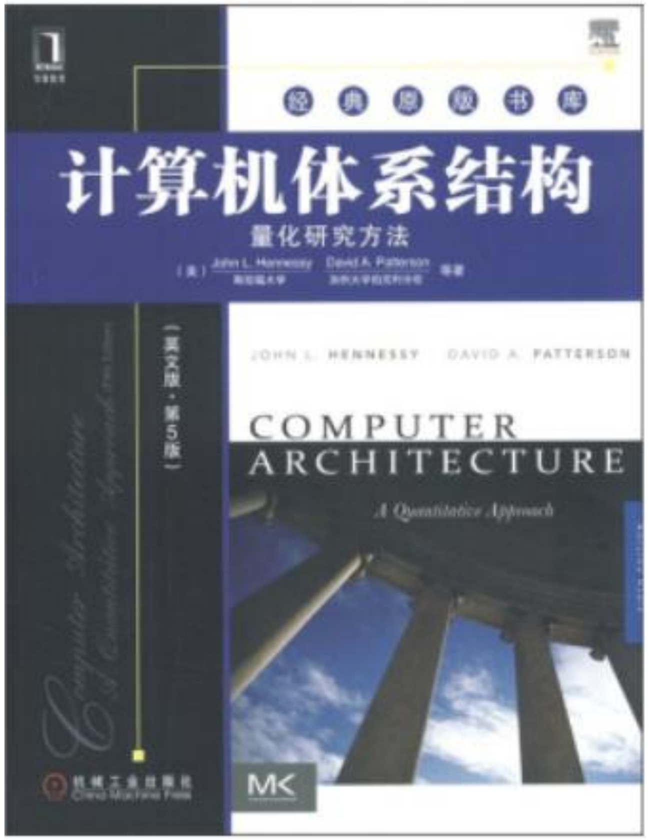



COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
RISC-V EDITION
MK
DAVID A. PATTERSON
JOHN L. HENNESSY

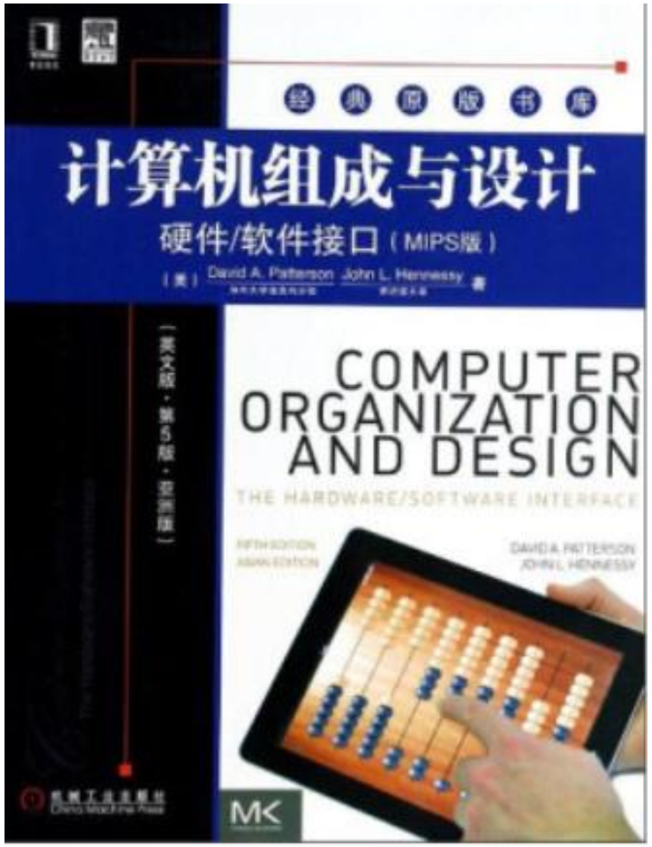
John L. Hennessy, David A. Patternson; Computer Architecture: A Quantitative Approach; sixth Edition.

David A. Patternson, John L. Hennessy ; Computer Organization and Design- The Hardware/Software Interface; RISC-V Edition.

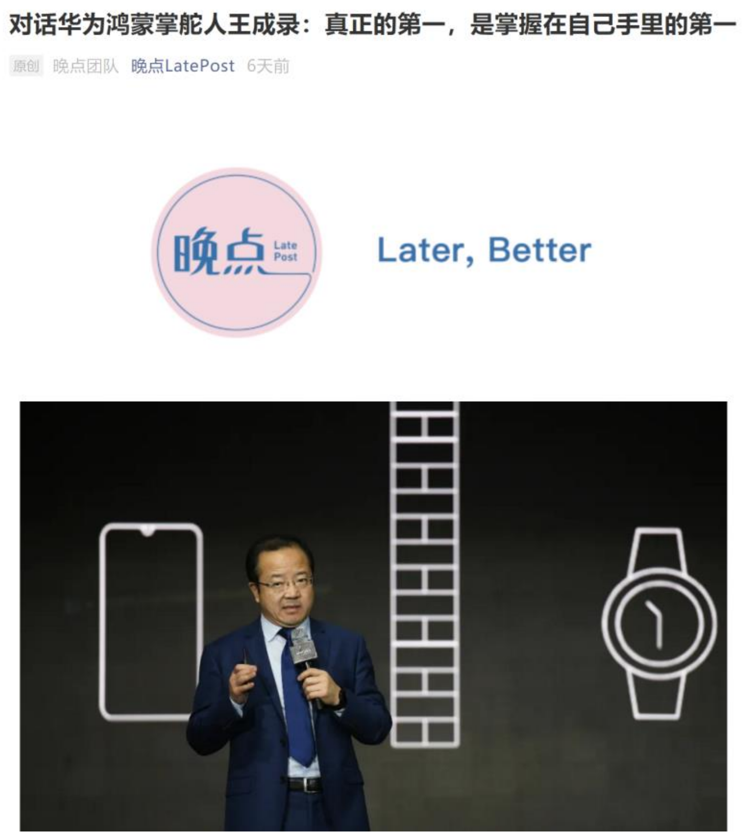
教材与主要参考书



John L. Hennessy, David A. Patternson；Computer Architecture: A Quantitative Approach. Fifth Edition. 机械工业出版社，2012

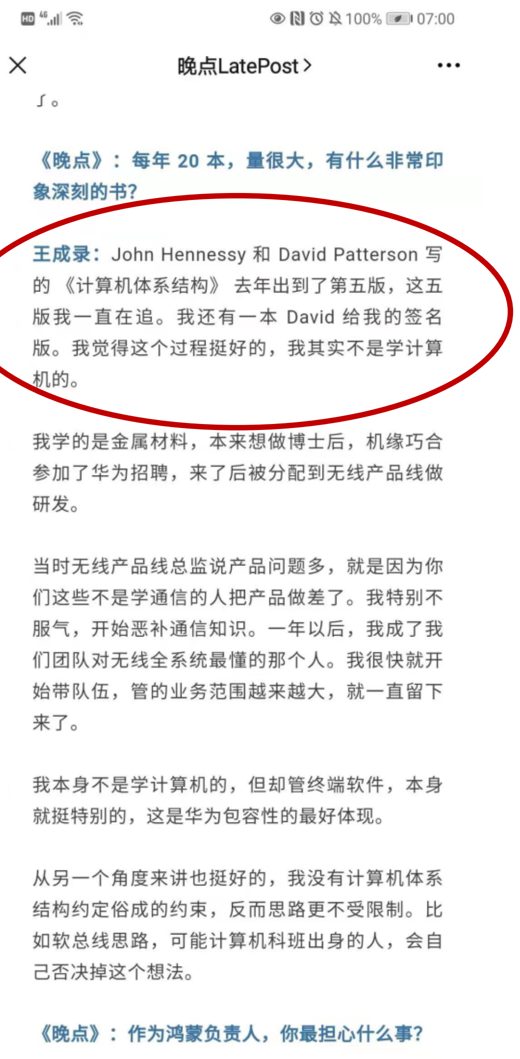


David A. Patternson, John L. Hennessy, Computer Organization & Design : The Hardware/Software Interface, Third Edition. San Francisco: Morgan Kaufmann Publishers, Inc. 2005



对话华为鸿蒙掌舵人王成录：真正的第一，是掌握在自己手里的第一

原创 晚点团队 晚点LatePost 6天前



」。

《晚点》：每年 20 本，量很大，有什么非常印象深刻的书?

王成录： John Hennessy 和 David Patterson 写的《计算机体系结构》去年出到了第五版，这五版我一直在追。我还有一本 David 给我的签名版。我觉得这个过程挺好的，我其实不是学计算机的。

我学的是金属材料，本来想做博士后，机缘巧合参加了华为招聘，来了后被分配到无线产品线做研发。

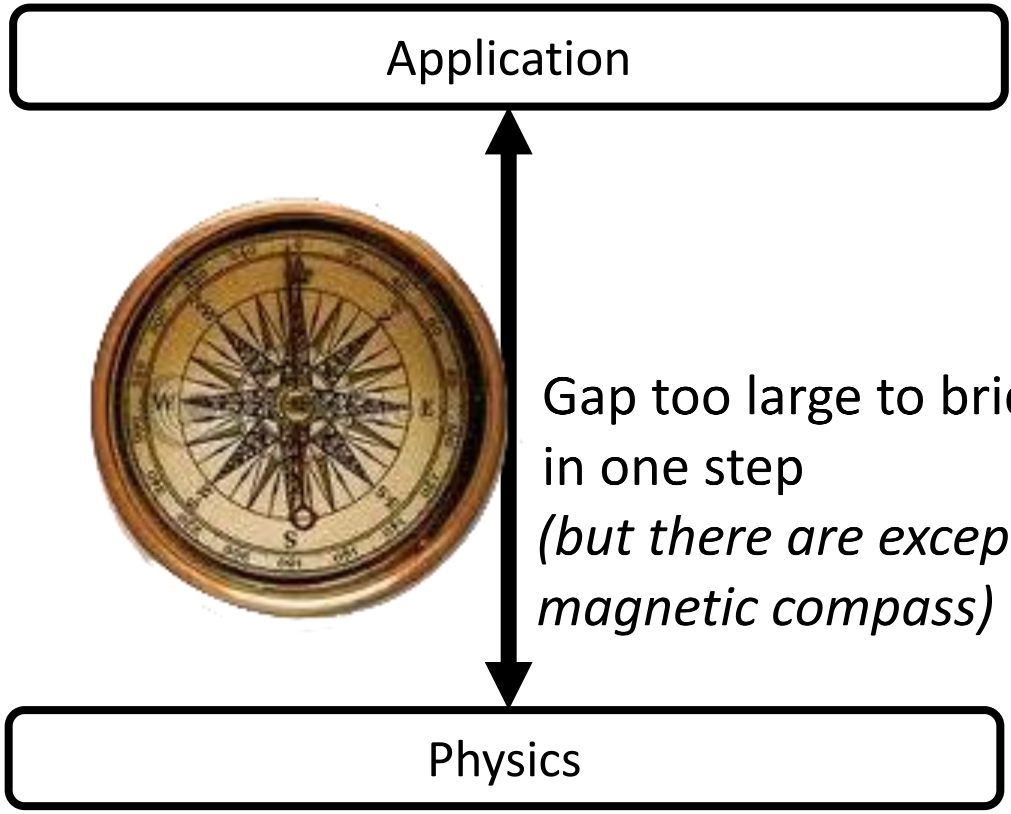
当时无线产品线总监说产品问题多，就是因为你们这些不是学通信的人把产品做差了。我特别不服气，开始恶补通信知识。一年以后，我成了我们团队对无线全系统最懂的那个人。我很快就开始带队伍，管的业务范围越来越大，就一直留下来了。

我本身不是学计算机的，但却管终端软件，本身就挺特别的，这是华为包容性的最好体现。

从另一个角度来讲也挺好的，我没有计算机体系结构约定俗成的约束，反而思路更不受限制。比如软总线思路，可能计算机科班出身的人，会自己否决掉这个想法。

《晚点》：作为鸿蒙负责人，你最担心什么事?

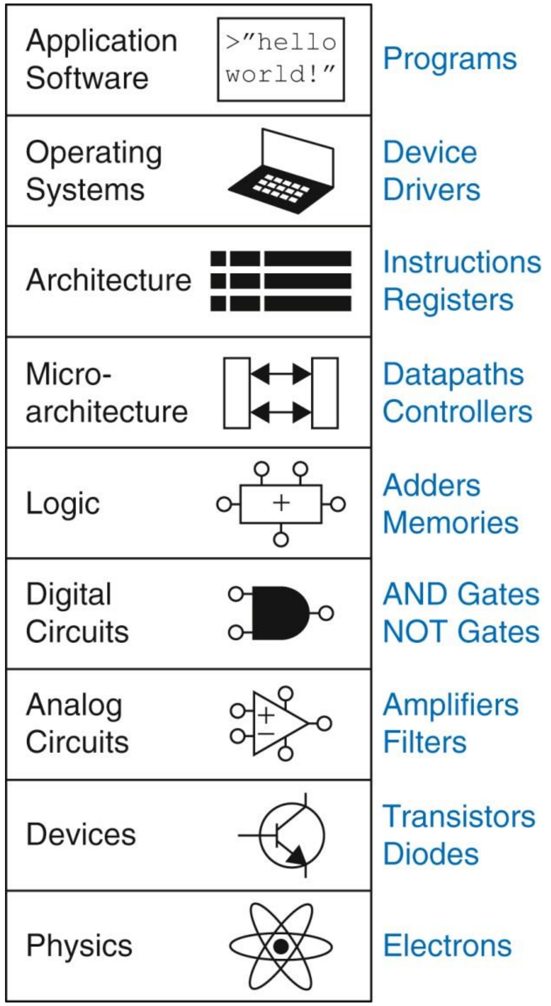
What is Computer Architecture?



广义的定义：计算机体系结构是抽象层的设计，这些抽象层使得我们可以使用可用的制造技术高效地实现信息处理应用系统

现代计算机系统的抽象层次

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machine
- Instruction Set Architecture (ISA)
- Microarchitecture
- Gates/Register-Transfer Level (RTL)
- Circuits
- Devices
- Physics



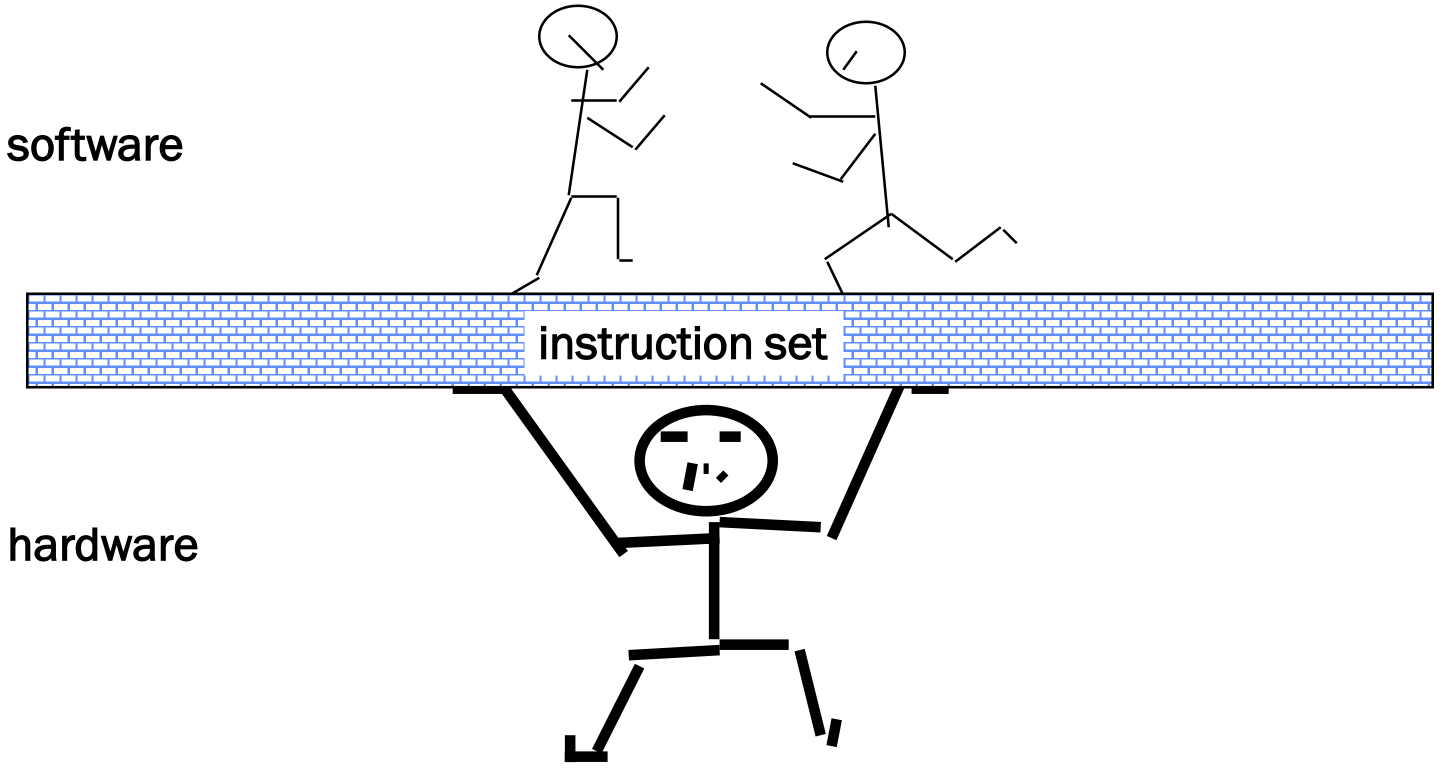
What is Computer Architecture?

- 计算机体系结构是研究如何选择（设计）功能部件和互联方法来满足计算机系统的功能、性能、价格约束的科学
- 描述计算机系统的功能、结构组织和实现的一组规则和方法。
- 计算机体系结构是软件设计者与硬件设备设计者（VLSI）之间的中间层，是软件与硬件的接口（Interface）
- 计算机体系结构定义为：一组指令及机器的一系列状态

计算机体系结构的定义（续）

- **过去的观点:**
  - 指令集架构 (ISA) 设计
  - 即 体系结构设计需要关注并确定:
    - 寄存器组织、存储模型、寻址方式、指令操作数、硬件支持的操作种类、指令编码方式
    - 如何处理中断和异常?
- **目前的观点:**
  - 根据目标机器的特定需求，在成本、功耗、可用性等约束下最大化机器性能
  - 包括 指令集架构（ISA）, 计算机组织（微体系结构）, 硬件实现

ISA: a Critical Interface



ISA需说明的主要内容

- Memory addressing
- Addressing modes
- Types and sizes of operands
- Operations
- Control flow instructions
- Encoding an ISA
- ……

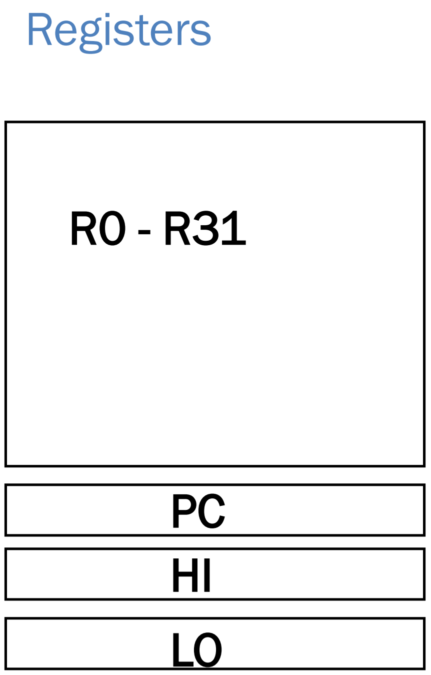
- 优秀的ISA所具有的特征
  - 可持续用于很多代机器上(portability)
  - 可以适用于多个领域 (generality)
  - 对上层提供方便的功能（convenient functionality)
  - 可以由下层有效地实现（efficient implementation)
  - ……

指令集结构举例

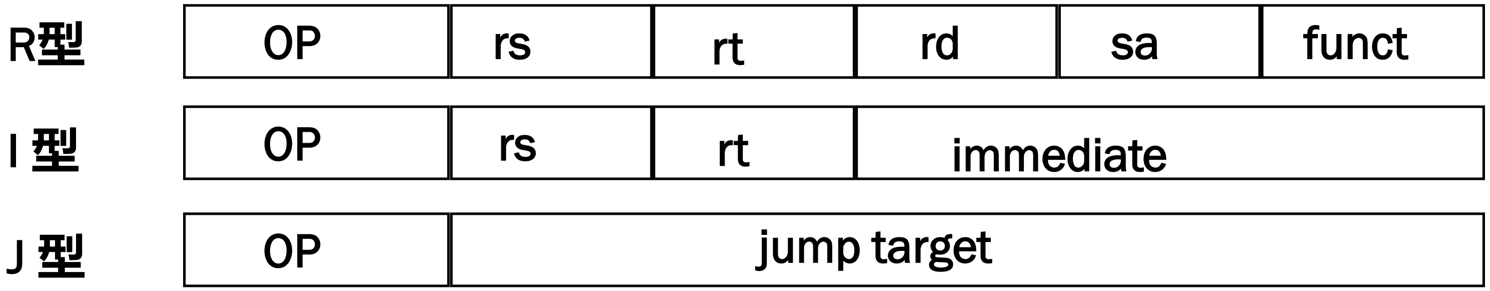
• Digital Alpha(v1, v3)	1992-97
• HP PA-RISC (v1.1, v2.0)	1986-96
• Sun Sparc(v8, v9)	1987-95
• SGI MIPS (MIPS I, II, III, IV, V)	1986-96
• Intel(8086,80286,80386, 80486,Pentium, MMX, ...)	1978-96

MIPS R3000 Instruction Set Architecture (Summary)

- **指令类型**
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



3 **种指令格式**: all 32 bits wide

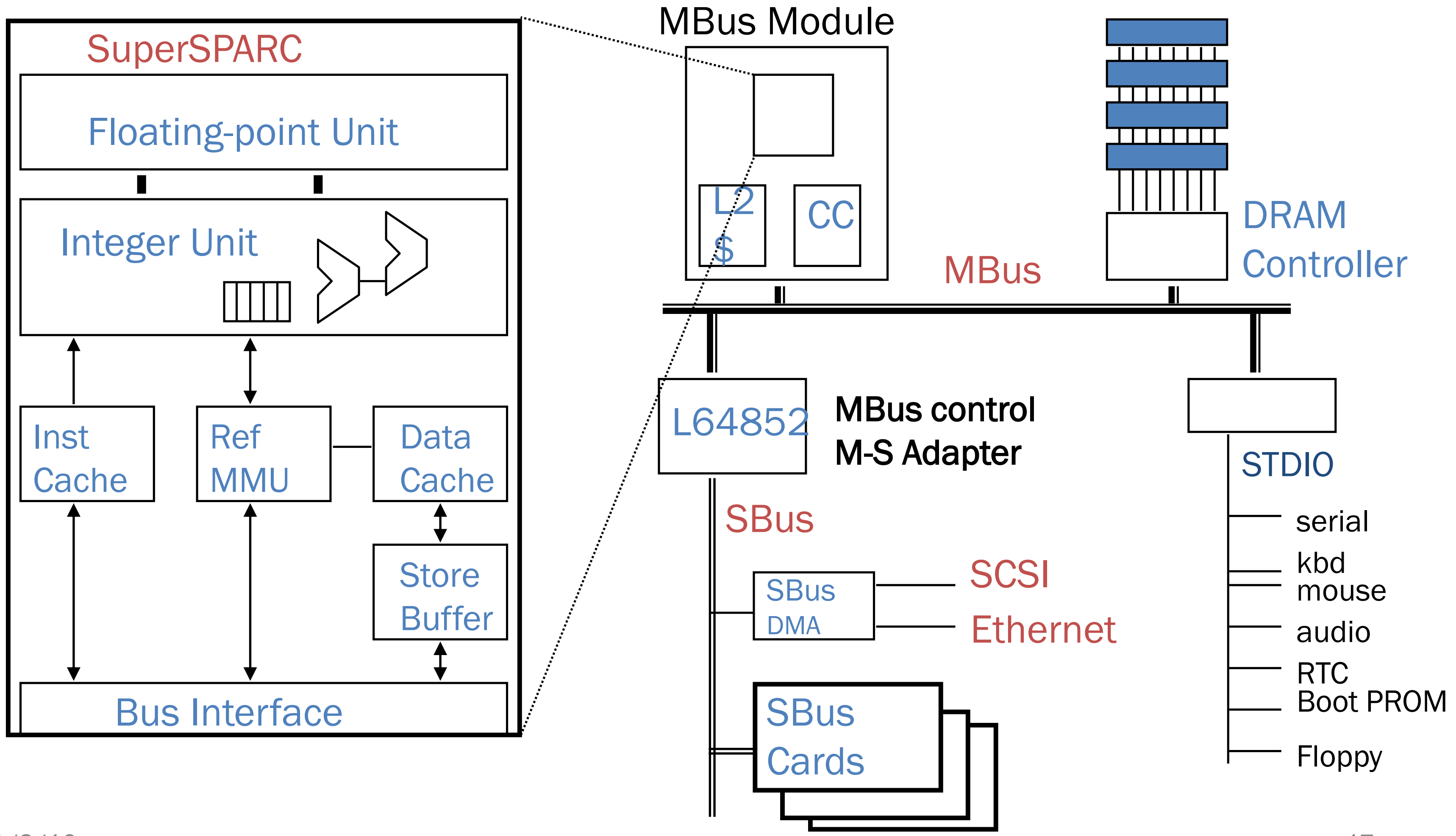


计算机组成与实现

- **计算机组成（Computer Organization or Microarchitecture): ISA的逻辑实现**
  - 物理机器级中的数据流和控制流的组成以及逻辑设计等
- **计算机实现（Computer Implementation):计算机组成的物理实现**
  - CPU， MEMORY等的物理结构，器件的集成度、速度，模块、插件、底板的划分与连接、信号传输、电源、冷却及整机装配技术等
- **例如**
  - 确定指令系统中是否有乘法指令 (Architecture)
  - 确定用加法器实现乘法 还是用专门的乘法实现 (Organization)
  - 器件的选定及所用的微组装技术 （Implementation)

Example Organization

- TI SuperSPARC^{tm} TMS390Z50 in Sun SPARCstation20



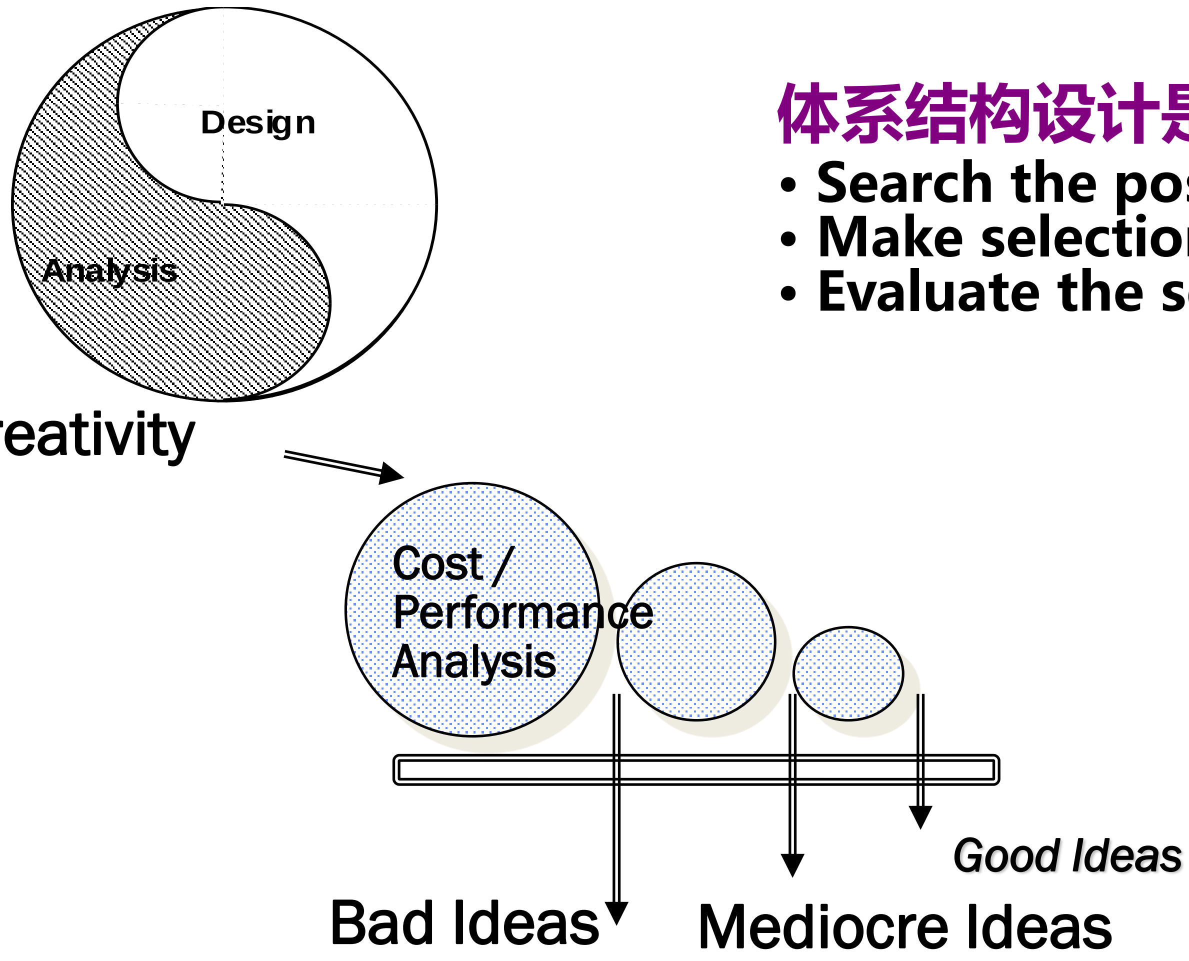
指令集架构 vs.微体系结构

- **Architecture / Instruction Set Architecture (ISA)**
  - Class of ISA: register-memory or register-register architectures
  - Programmer visible state (Register and Memory)
  - Addressing Modes: how memory addresses are computed
  - Data types and sizes for integer and floating-point operands
  - Instructions, encoding, and operation
  - Exception and Interrupt semantics
- **Microarchitecture / Organization**
  - Tradeoffs on how to implement the ISA for speed, energy, cost
  - Pipeline width and depth, cache size, peak power, bus width, execution order, etc

计算机体系结构设计者的任务

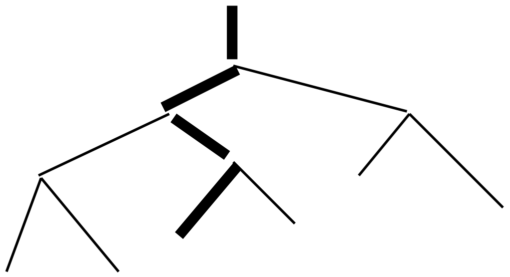
- **设计和实现不同档次的计算机系统**
  - Understand software demands
  - Understand technology trends
  - Understand architecture trends
  - Understand economics of computer systems
- **最大化性能、可编程性等指标**
  - 在一定的技术和成本的限制下
- **体系结构现状:**
  - 现代微处理器大多为多核处理器
  - 单芯片中通常集成多个处理器核心
  - 每个处理器核心支持多线程执行

计算机体系结构设计过程



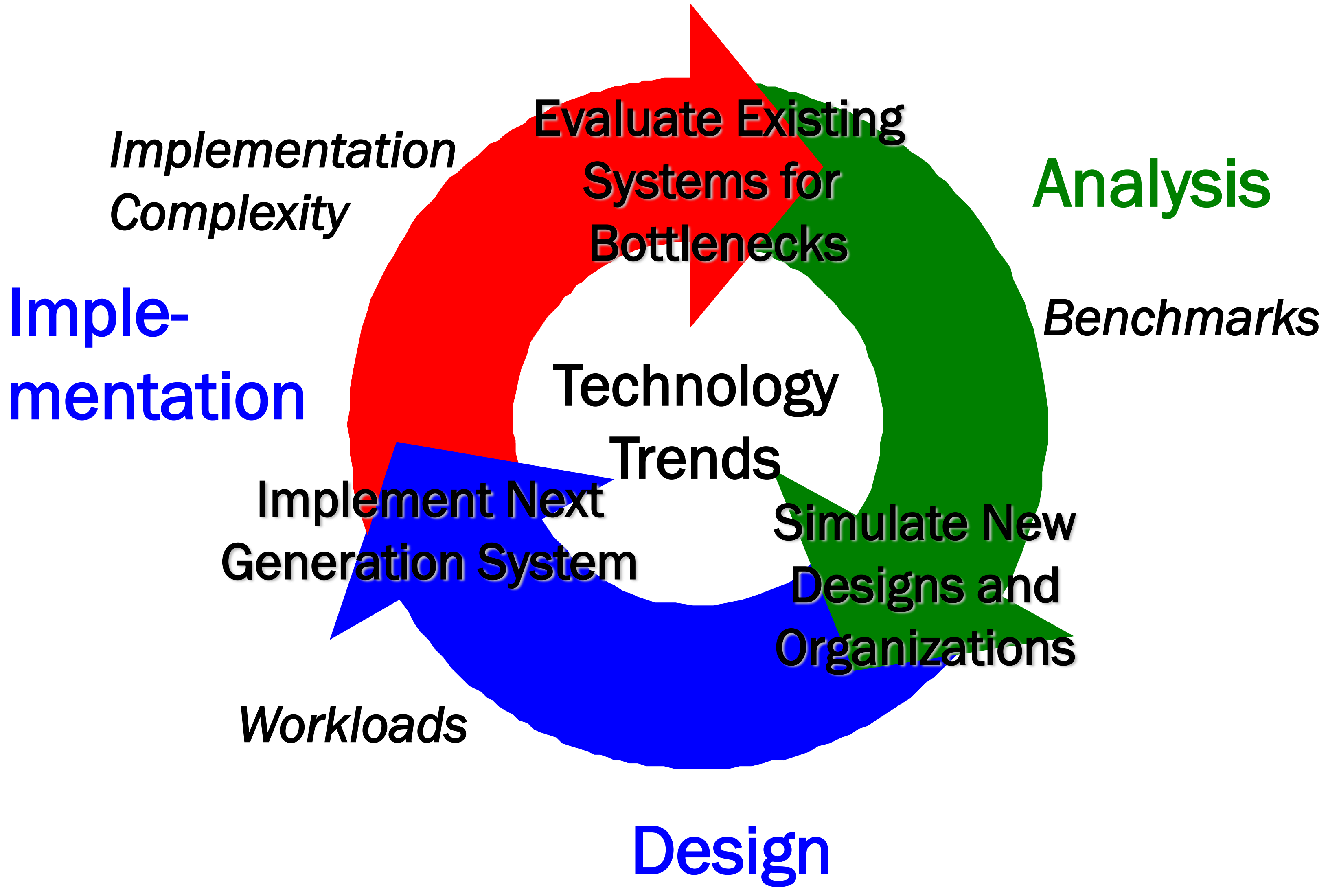
体系结构设计是循环渐进的过程:

- Search the possible design space
- Make selections
- Evaluate the selections made



Good measurement tools are required to accurately evaluate the selection.

计算机工程方法学


Evaluate Existing Systems for Bottlenecks
Analysis
Benchmarks
Implementation Complexity
Imple-mentation
Technology Trends
Implement Next Generation System
Simulate New Designs and Organizations
Workloads
Design

课程目标

- 掌握系统定量分析的基本方法和技术
- 深入理解提高CPU性能的基本方法
- 深入理解存储系统的基本原理和优化方法
- 理解数据级并行、指令级并行、线程级并行的基本原理和方法
- 初步理解面向特定领域的处理器设计

本课程的主要内容

- **简单机器设计（Chapter 1, Appendix A, Appendix C)**
  – ISAs, Iron Law, simple pipelines
- **存储系统(Chapter 2，Appendix B)**
  – DRAM, caches, virtual memory systems
- **指令级并行(Chapter 3)**
  – score-boarding, out-of-order issue
- **数据级并行(Chapter 4)**
  – vector machines, VLIW machines, multithreaded machines
- **线程级并行(Chapter 5)**
  – memory models, cache coherence, synchronization
- **面向特定领域的处理器体系结构（DSA)**
  – IPU、DSP、GPU

为什么学这门课

深入理解计算机体系结构:

- **开展体系结构研究与设计的基础**
  - 体系结构领域仍然存在许多挑战性问题
    - 例如：CPU与memory之间性能差异
    - 功耗和能耗问题
    - 体系结构与应用特征的适配性问题等
  - …….
- **更好地设计与实现操作系统、编译器**
  - 需要重新评估当前的假设和权衡
    - 例如: 当面临网络性能持续提升、并行系统、异构系统日益普遍
  - 现代计算机需要更好的优化编译器和更好的编程语言
- **更好地设计与实现应用程序**
  - 可更好地理解算法、数据结构和编程语言选择对性能的影响

评分规则

- **授课**
  - 授课总学时60学时，实验30学时
  - 3C301: 1(6,7)，3(6,7)
- **评分**
  - 平时作业 10%
  - 随堂测验 15%
  - 实验 40%
  - 期终考试 35%

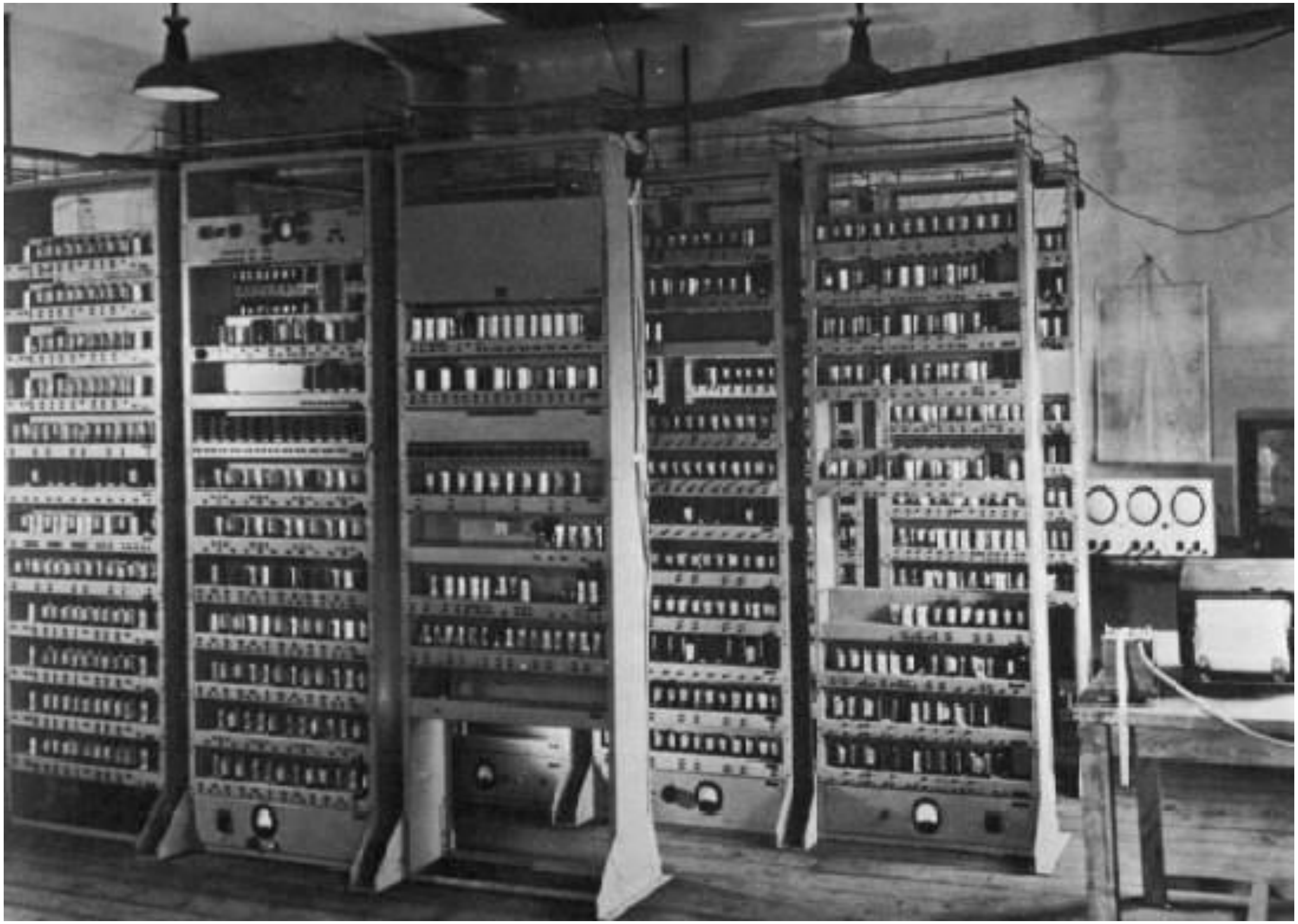
关于作弊

- 作业
- 实验
- 考试（测验）
- 一旦检查出来作弊，这门课不及格

Chapter1 量化设计与分析基础

- **1.1 引言**
  - 计算机体系结构的定义
  - 计算机的分类
  - 现代计算机系统发展趋势
- **1.2 定量分析基础**

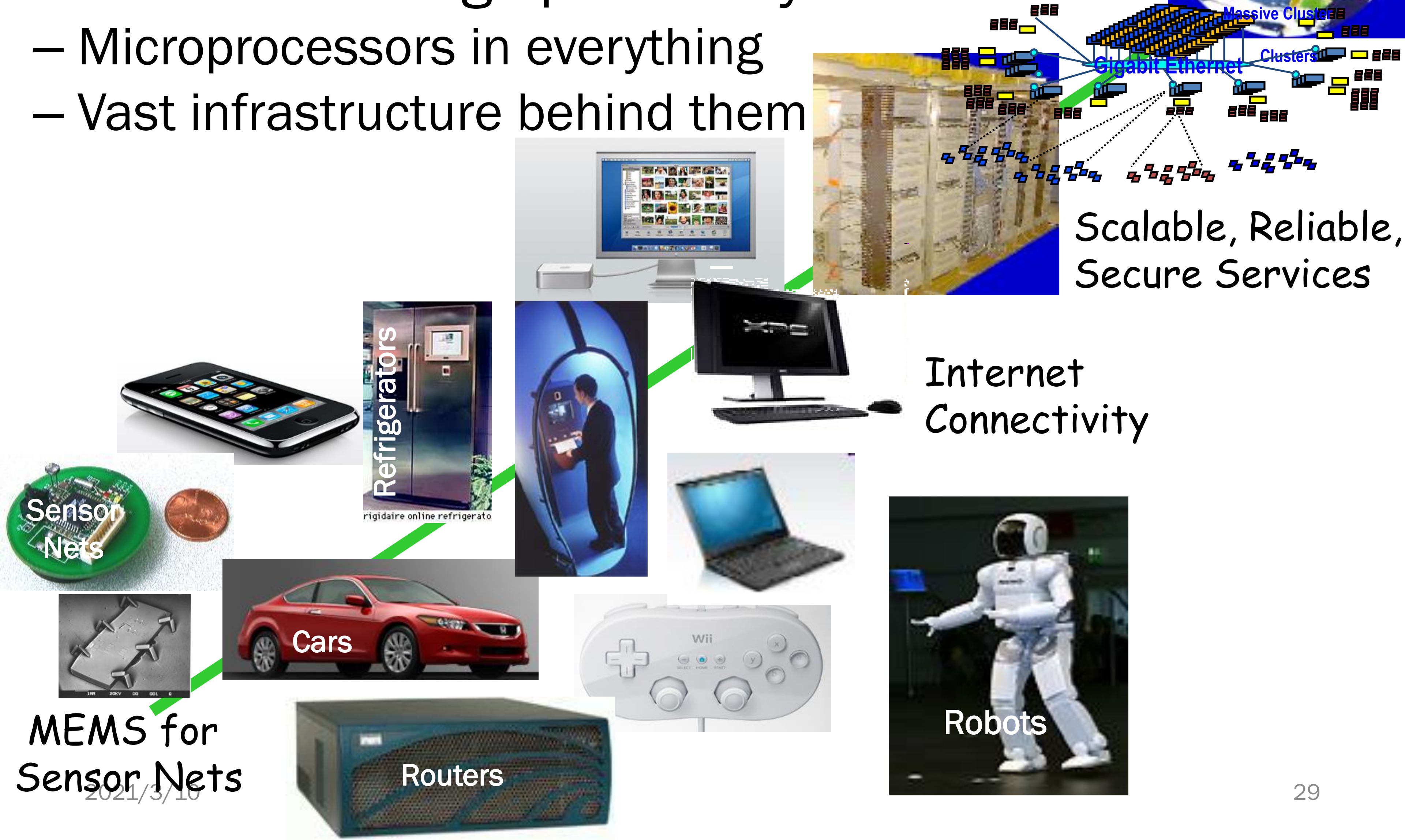
Computing Devices Then…



EDSAC, University of Cambridge, UK, 1949
EDSAC, Electronic Delay Storage Automatic Calculator

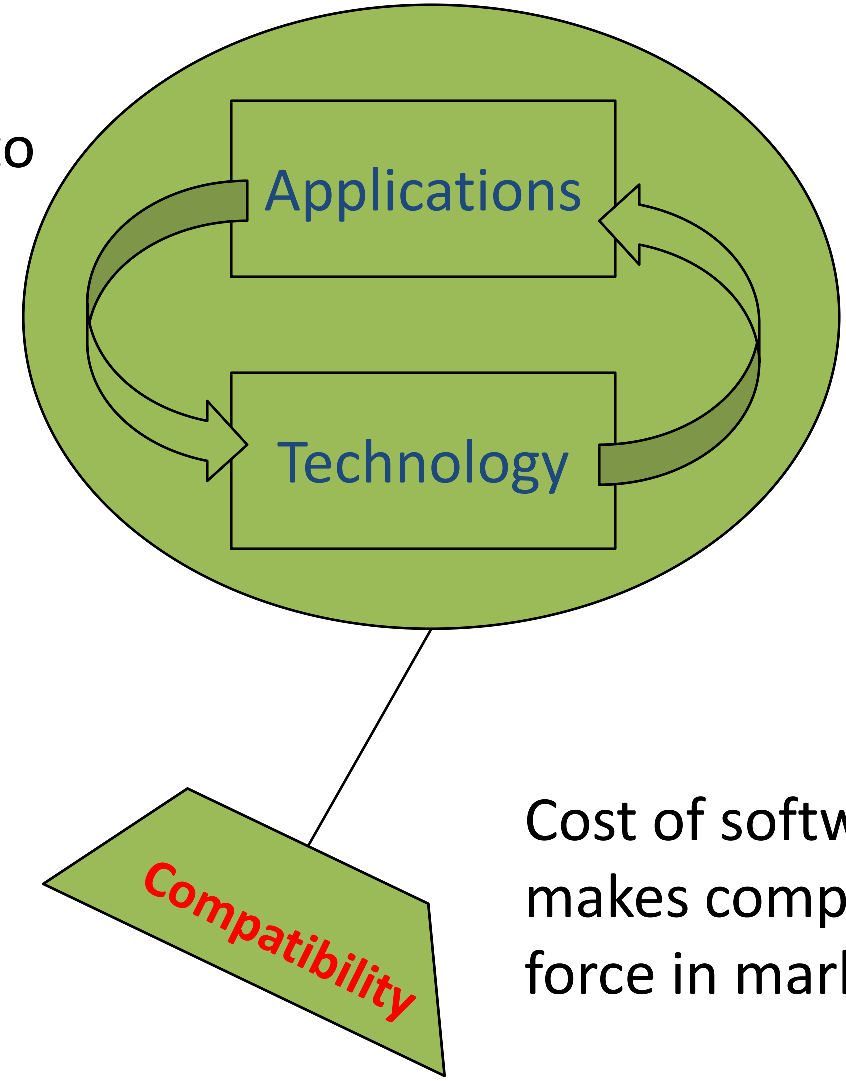
Computing Systems Today

The world is a large parallel system
- Microprocessors in everything
- Vast infrastructure behind them



体系结构发展的驱动力

Applications suggest how to improve technology, provide revenue to fund development



Improved technologies make new applications possible

Cost of software development makes compatibility a major force in market

体系结构的发展历史、现状及趋势*

*A New Golden Age for Computer Architecture: Domain-Specific Hardware/Software Co-Design, Enhanced Security, Open Instruction Sets, and Agile Chip Development

John Hennessy and David Patterson
June 4, 2018
https://www.youtube.com/watch?v=3LVeEjsn8Ts
https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

- **体系结构发展历史**
  – Mainframes

- as transistor density increased, power consumption per transistor would drop
- Voltage and current proportional to the transistor area
- Ignored leakage current and threshold voltage
- Power wall

- ore’s Law 的终结
  – Security

The number of transistors per chip, at constant cost, doubles every 18-24 months

- **体系结构发展机遇**
  – Open Architectures/open ISA (e.g., RISC-V)
  – Domain Specific Languages and Architecture
  – Agile Hardware Development

体系结构发展的机遇*

- **A New Golden Age for Computer Architecture**
  - By Hennessy and Patterson, June 4, 2018, the 45th ISCA
  - https://www.acm.org/hennessy-patterson-turing-lecture
  - Open Architectures
  - Domain Specific Architectures
  - Agile Hardware Development
  - Enhancing Security

体系结构发展的机遇*

- **Open Architectures**
  - 软件技术的进步激发体系结构创新
  - 为什么有开放的编译器、操作系统而没有开放的ISA
  - 现阶段，使用特定的ISA就要给设计方交版权费，设计方很少公开他们做某个选择的原因

RISC V 第五代Berkeley RISC

体系结构发展的机遇*

- **RISC V**
  - A practical ISA that is open-sourced
  - 很多元素来自学校的“处理器设计”相关projects
  - 拟支持嵌入式、PC、Supercomputer、向量机等
  - You can join! https://riscv.org/membership-application/

体系结构发展的机遇*

- **Domain Specific Languages and Architecture**
  - 提高性能的路径：Domain Specific Architectures(DSAs)
  - 根据应用特征调整体系结构来实现更高的效率
    - 对于特定的领域，更有效的挖掘计算并行性
    - 对于特定的领域，更有效的利用内存带宽
    - 消除不必要的精度
  - 并不是仅针对一个专门的应用, 而是通过执行软件适应某一领域。
    - 例如机器学习芯片: 寒武纪芯片、TPU芯片

体系结构发展的机遇*

- **Agile Hardware Development**
  - Agile development: It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change
  - Agile software development is a norm
  - Is agile hardware development possible?

Small programming teams quickly developed working-but-incomplete prototypes and got customer feedback before starting the next iteration

20世纪60年代初IBM机器兼容性问题

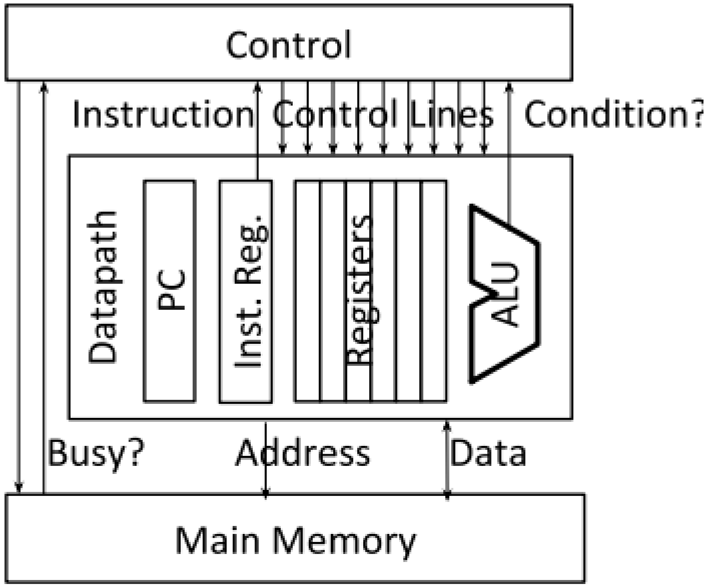
- **20世纪60年代初，IBM有4条产品线**
  - 701 ->7094
  - 650 ->7074
  - 702 -> 7080
  - 1401->7010
- **每个系统包含各自不同的**
  - ISA
  - I/O 系统及二级存储：磁带、磁鼓和磁盘
  - 汇编器、编译器、库等
  - 市场定位：商业应用、科学计算、实时系统

系列机：IBM System/360 – one ISA to rule them all

The **IBM System/360** (**S/360**) is a family of mainframe computer systems that was announced by IBM on April 7, 1964, and delivered between 1965 and 1978.[1] It was the first family of computers designed to cover the complete range of applications, from small to large, both commercial and scientific. The design made a clear distinction between architecture and implementation, allowing IBM to release a suite of compatible designs at different prices. All but the only partially

数据通路vs控制

- 处理器设计可划分为两部分
  - Datapath： 存储和操作数据
  - Control：产生控制信号作用于数据通路
- 过去对处理器设计师的最大的挑战是产生正确的控制序列



* "Micro-programming and the design of
M. Wilkes, and J. Stringer. *Mathematic*
1953.

- Maurice Wilkes **发明了微程序设计方法来设计控制部件*** （1953）

  - Stored program vs circuitry
  - Logic expensive vs. ROM or RAM
  - ROM cheaper than RAM
  - ROM much faster than RAM

Microprogramming, Process of writing microcode for a microprocessor. Microcode is low-level code that defines how a microprocessor should function when it executes machine-language instructions. Typically, one machine-language instruction translates into several microcode instructions. On some computers, the microcode is stored in ROM and cannot be modified; on some larger computers, it is stored in EPROM and therefore can be replaced with newer versions.

IBM 360系列机的控制部件

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000

Fred Brooks, Jr.

Frederick Phillips "Fred" Brooks Jr. (born April 19, 1931) is an American computer architect, software engineer, and computer scientist, best known for managing the development of IBM's System/360 family of computers and the OS/360

IC技术、微程序技术催生CISC

- **逻辑电路、RAM和ROM使用同样的晶体管实现**
- **半导体RAM的速度与ROM的速度基本相同**
- **控制存储密度会持续增长（Moores Law)**
- **由于采用RAM存储控制信号，容易修复微代码bug**

- **综上允许更加复杂的ISAs**
- **CISC: Complex Instruction Set Computers**
- **例如：小型机（TTL server)**
  - Digital Equipment Corp. (DEC)
  - VAX ISA in 1977
  - 5K × 96b 微码

Writable Control Store

- **如果控制存储是RAM，那么就可以定制“固件”应用程序：“Writable Control Store”**
- **微程序研究在学术界很流行**
  - Patterson Phd Thesis*
  - 有专门的国际会议SIGMICRO
- **Xerox Alto （Bit Slice TTL） （1973）**
  - 第1台具有GUI和网络的个人计算机
  - BitBlt和网络控制器用微码实现

Chuck Thacker

* *Verification of microprograms*, David Patterson, UCLA, 1976
** “The design of a system for the synthesis of correct microprograms,” David Patterson, *Proc. 8th Annual Workshop of Microprogramming*, 1975

微处理器演进

- **上世纪70年代，在MOS技术进步的推动下，小型计算机和主机ISAs得到了迅速发展**
- **“Microprocessor Wars”：通过添加指令，调整汇编器**
- **Intel iAPX 432：最具雄心的微型计算机始于1975年**
  - 基于32位能力的面向对象体系结构，使用Ada编写的定制操作系统
  - 严重的性能、复杂性(多芯片)和可用性问题导致1981年发布
- **Intel 8086 (1978, 8MHz, 29,000 transistors)**
  - 要求：“Stopgap” 16-bit processor, 52 周开发新的芯片
  - 结果：10人3周开发了汇编级兼容8080的ISA
- **1981年IBM 采用Intel8088 （8位数据总线）研制了IBM PC机**
  - 希望1986年 销售25万台，实际销售1+亿台
  - PC软件的二进制兼容=> 8086前途光明

80年初：微程序控制机器分析

- **用高级语言编程成为主流**
  - 关键问题：编译器会生成什么指令？(ISA vs. Compiler)
- **IBM的John Cocke团队**
  - 为小型计算机801（ECL Server）开发了更简单的 ISA 和编译器
  - 移植到IBM370，仅使用IBM 370的简单的寄存器-寄存器及load/store指令
  - 发现：与原IBM 370相比，性能提高3X
- **80年代初，Emer和Clark （DEC）发现**
  - VAX 11/180 CPI = 10!
  - VAX ISA 的 20%指令 （占用了60%的微码）仅占用了 0.2%的执行时间
- **Patterson：如何修复微处理器中的微程序bug，投稿’79DEC 后，引发对ISA合理性的研究**

* "A Characterization of Processor Performance in the VAX-11/780," J. Emer and D.Clark, *ISCA*, 1984.
** "RISCy History," David Patterson, May 30, 2018, Computer Architecture Today Blog

From CISC to RISC

- **CISC指令系统主要存在以下几方面的问题：**
  - 指令使用频度；CISC指令系统复杂-〉增加研制时间和成本，容易出错；
  - VLSI设计困难，不利于单片集成；
  - 许多复杂指令操作复杂，运行速度慢；
  - 各条指令不规整，不利于先进计算机体系结构技术来提高系统的性能。
- **使用简单ISA**
  - 指令像微指令那样简单
  - 编译生成的指令仅是部分CISC指令
  - 能够采用流水线方式实现
- **技术基础**
  - 使用指令Cache
  - 芯片集成度有很大提高：80年代初，单芯片已经可以集成32位数据通路+小规模cache
  - Chaitin的寄存器分配方法有利于Load-store型ISA

*Chaitin, Gregory J., et al. "Register allocation via coloring." *Computer languages* 6.1 (1981), 47-57

CISC vs. RISC Today

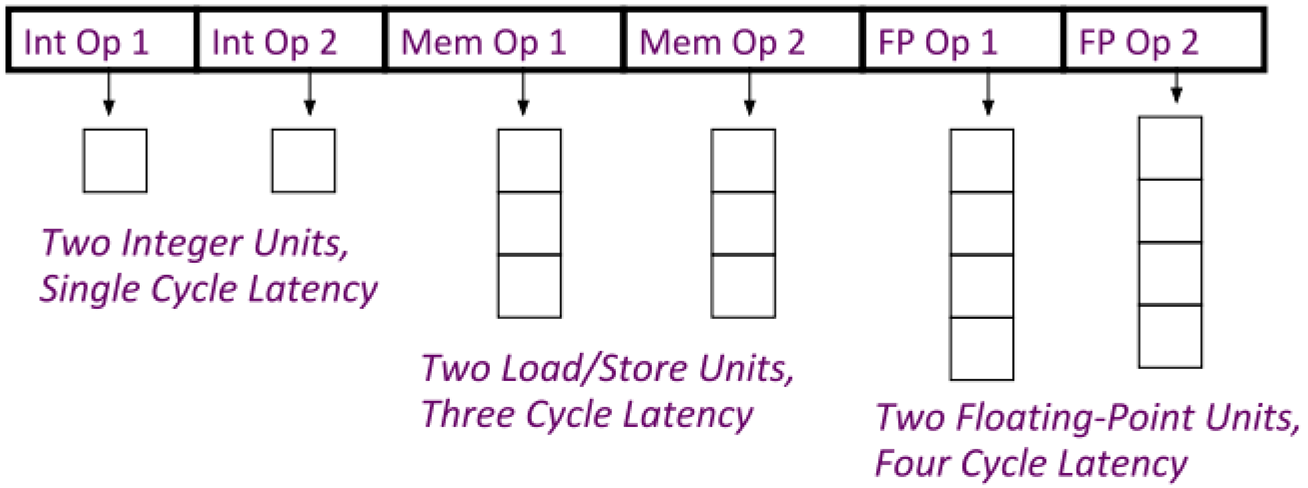
PC 时代

- 硬件将x86指令翻译为内部的RISC 指令
- 在MPU内部使用RISC技术
- x86 ISA在桌面和服务器市场占主导地位
- >350M / year

后PC时代: Client/Cloud

- SoC中的处理器核 vs. MPU
- 芯片面积、能耗和性能同样重要
- 99%的处理器是RISC
- >20B total/year in 2017
  - x86在2011年达到顶峰，现在每年下降~8%，
  - x86服务器=>Cloud ~10M servers
  - Total (0.05% of 20B)

VLIW: Very Long Instruction Word


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

- 一条指令中包含多个操作
- 每个指令槽表示固定的功能
- 指定了恒定的操作延时
- 体系结构必须保证
  - 指令中的操作是并行的 => no cross-operation RAW check
  - 数据未准备好不使用数据 => no data interlocks

From RISC to Intel/HP Itanium，EPIC IA-64

- **EPIC 是Intel为他们的VLIW结构的命名**
  - Explicitly Parallel Instruction Computing
  - 二进制 目标码兼容的 VLIW
  - 从1994年与HP合作开发
- **EPIC IA-64 是 Intel 32位x86的后继（64位ISA)**
  - IA-64= Intel Architecture 64-bit
  - AMD 有自己的AMD64 技术,2003年推出业界首款64位处理器
  - 第一款Itanium 2002年推出，不兼容IA-32
  - 很多公司放弃RISC转而选择Itanium，因为他们普遍认为这是必然的（Microsoft, SGI, Hitachi,...

VLIW的问题及EPIC的失败

- 编译器无法处理整型类代码(指针)中的复杂依赖项
- 代码量膨胀
- **不可预知的分支**
- **可变的存储访问延迟（不可预知的cache失效）**
  - 乱序执行技术可处理Cache延迟
- **乱序执行覆盖了VLIW的优势**
- *The Itanium approach...was supposed to be so terrific –until it turned out that the wished-for compilers were basically impossible to write."*
  - Donald Knuth, Stanford

小结：体系结构发展历史

- **重要事件**
  - IBM 系列机、微程序设计、RISC、VLIW、EPIC
- **30年来没有出现新的通用CISC ISA**
- **15年来没有出现新的通用VLIW**
  - 在通用计算领域VLIW是失败的
  - 复杂的VLIW结构接近顺序超标量结构，但在大型复杂应用中不存在优势
  - VLIW在嵌入式DSP市场比较成功
    - 简单VLIW，分支简单、没有Cache，小程序
- **RISC！普遍认为RISC原则是通用ISA的最好原则**

小结：体系结构发展历史

- **重要事件**
  - IBM 系列机、微程序设计、RISC、VLIW、EPIC
- **30年来没有出现新的通用CISC ISA**
- **15年来没有出现新的通用VLIW**
  - 在通用计算领域VLIW是失败的
  - 复杂的VLIW结构接近顺序超标量结构，但在大型复杂应用中不存在优势
  - VLIW在嵌入式DSP市场比较成功
    - 简单VLIW，分支简单、没有Cache，小程序
- **RISC！普遍认为RISC原则是通用ISA的最好原则**

Eight Great Ideas in Computer Architecture

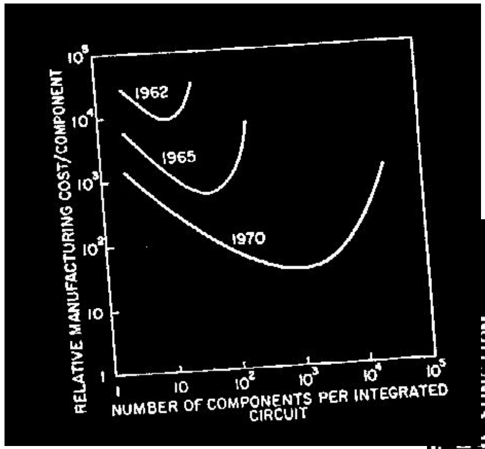
These are eight great ideas that computer architects have invented in the last 60 years of computer design. They are so powerful they have lasted long after the first computer that used them, with newer architects demonstrating their admiration by imitating their predecessors -- Patterson

https://www.elsevier.com/connect/8-great-ideas-in-computer-architecture

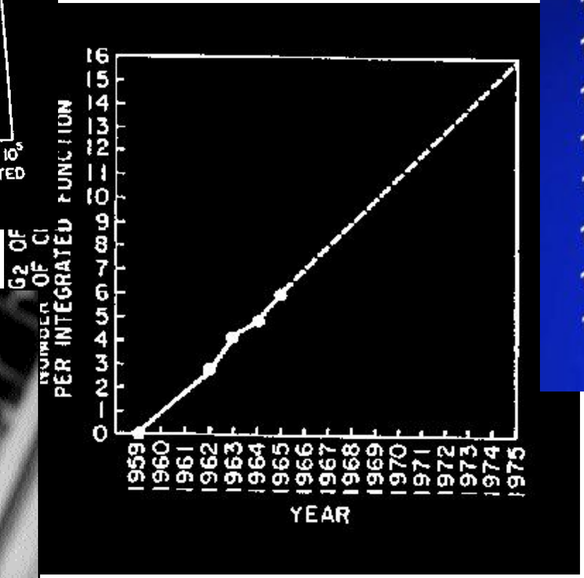
Eight Great ideas in Computer Architecture

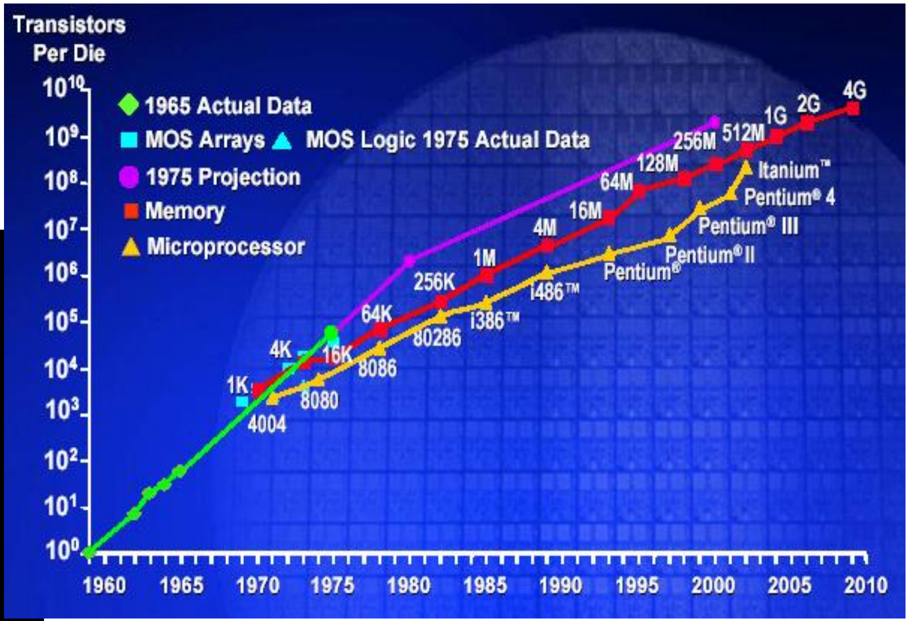
1. **Design for Moore’s Law**
2. **Abstraction to Simplify Design**
3. **Make the Common Case Fast**
4. **Dependability via Redundancy**
5. **Memory Hierarchy**
6. **Performance via Parallelism**
7. **Performance via Pipelining**
8. **Performance via Prediction**

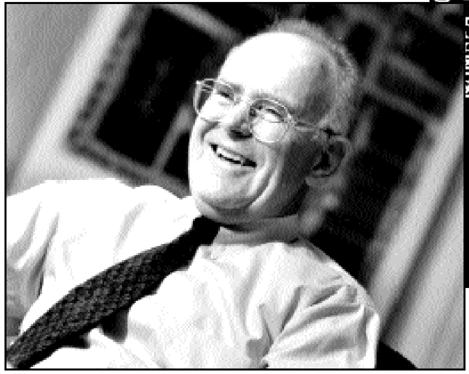
1. Moore’s Law（摩尔定律）



Who is Moore?







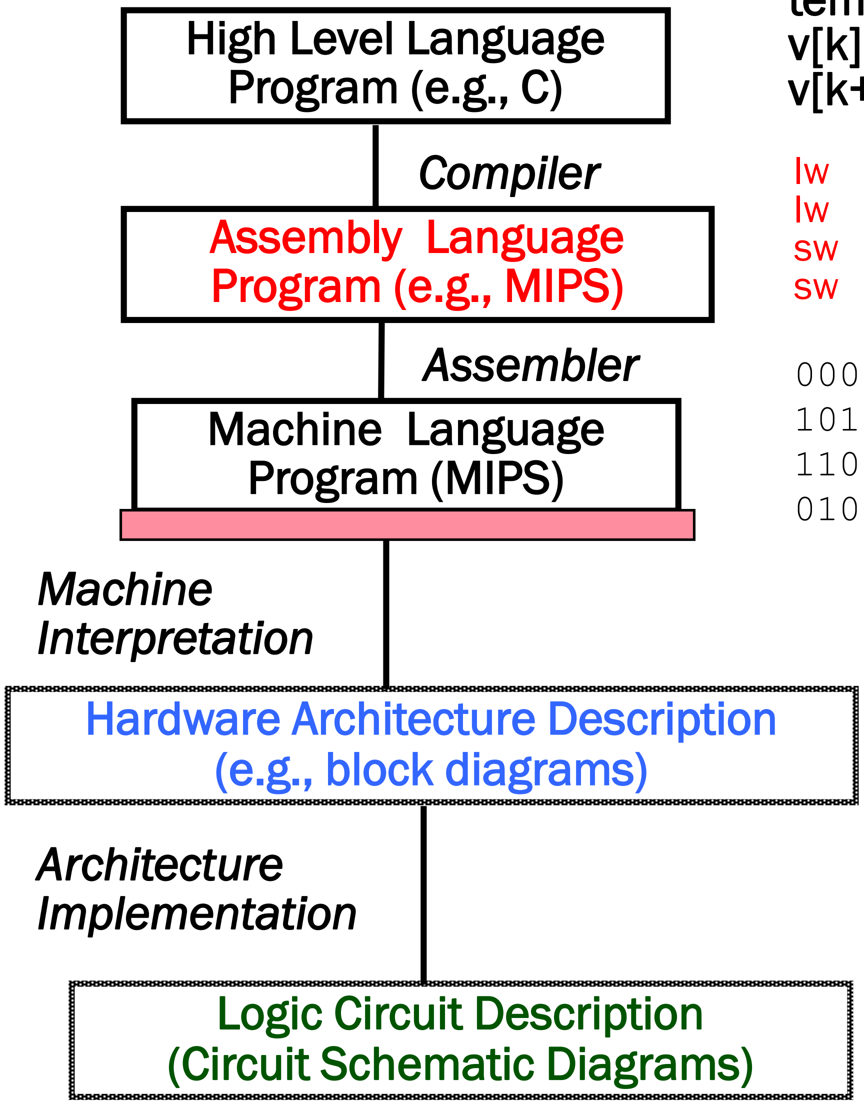
1. 集成电路芯片上所集成的电路的数目，每隔18个月就翻一番。
2. 微处理器的性能每隔18个月提高一倍，或价格下降一半。
3. 用一个美元所能买到的电脑性能，每隔18个月翻两番。

- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every 18 months

1. Design for Moore’ s Law

The one constant for computer designers is rapid change, which is driven largely by Moore's Law. It states that integrated circuit resources double every 18–24 months. Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel. As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project. Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

2. Abstraction via Layers of Representation


High Level Language Program (e.g., C)
Compiler
Assembly Language Program (e.g., MIPS)
Assembler
Machine Language Program (MIPS)
Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)
Architecture Implementation
Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Abstraction uses multiple levels with each level hiding the details of levels below it. For example:

The instruction set of a processor hides the details of the activities involved in executing an instruction.

High-level languages hide the details of the sequence of instructions needed to accomplish a task.

Operating systems hide the details involved in handling input and output devices.

2. Use Abstraction to Simplify Design

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law. A major productivity technique for hardware and soft ware is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels. We'll use the abstract painting icon to represent this second great idea.

GIVE ME AN EXAMPLE?

3. Make the Common Case Fast

- 在进行设计选择时，注重优化经常发生的事件
- 优化不经常执行的代码意义不大
- 选择一种（性能）度量方式来确定经常性事件 (Common case)

你的任务是把人以最快速度从A点送到B点，绝大多数情况下只有一个人。你选哪辆车？

在设计ISA时该如何利用这个idea?

如何知道哪个指令最常运行?

4. Dependability via Redundancy

- 通过冗余使得部分部件失效不影响整个系统的运行

假设你的预算为T，是买一台单价为T的大容量磁盘，还是买100台单价为T/100的廉价磁盘？

Storage servers with 24 hard disk drives and built-in hardware RAID controllers supporting various RAID levels

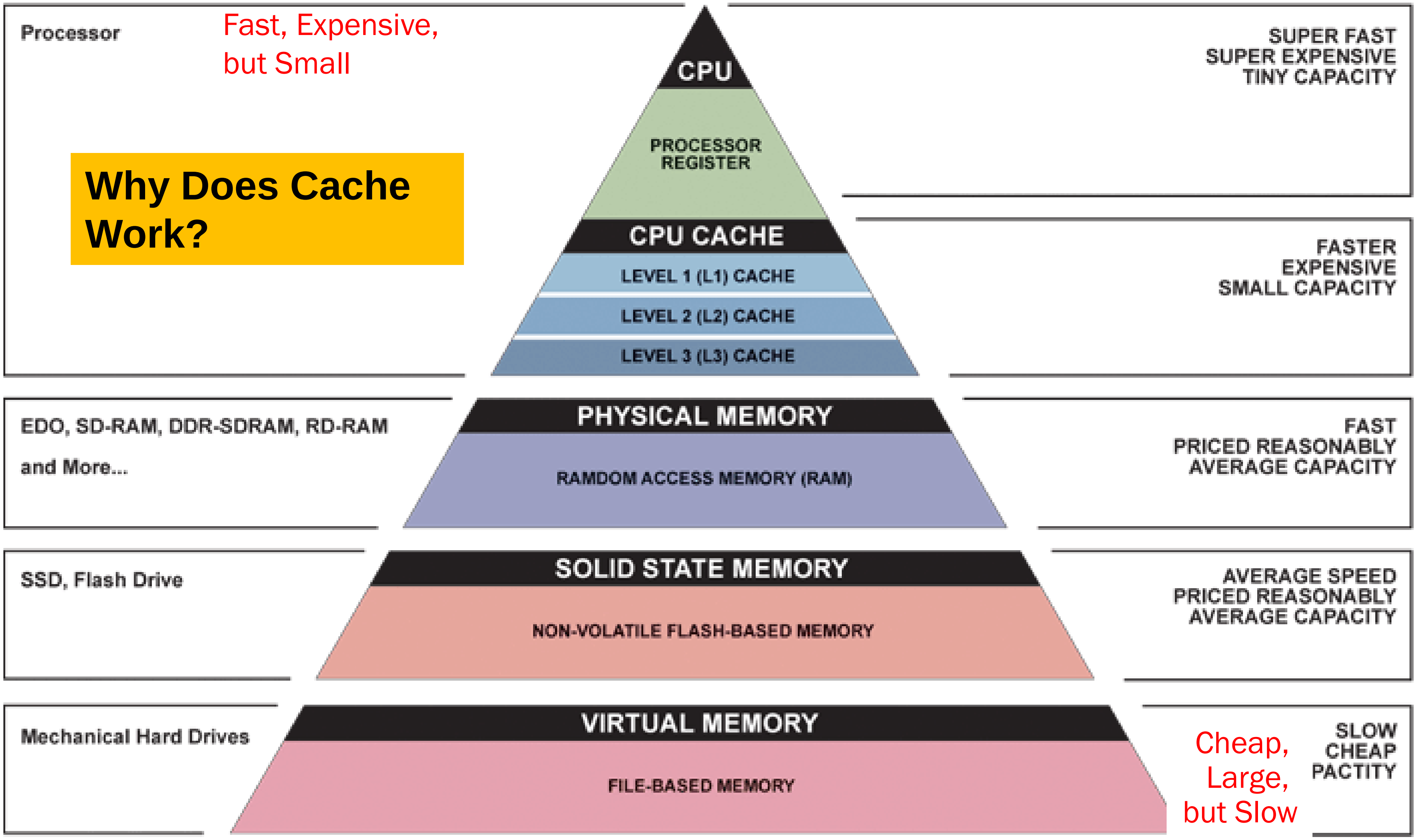
One of the most important ideas in data storage is the Redundant Array of Inexpensive Disks (RAID) concept. In most versions of RAID, data is stored redundantly on multiple disks. The redundancy insures that if one disk fails the data can be recovered from other disks.

5. Memory Hierarchy

Fast, Expensive, but Small

Why Does Cache Work?

Cheap, Large, but Slow



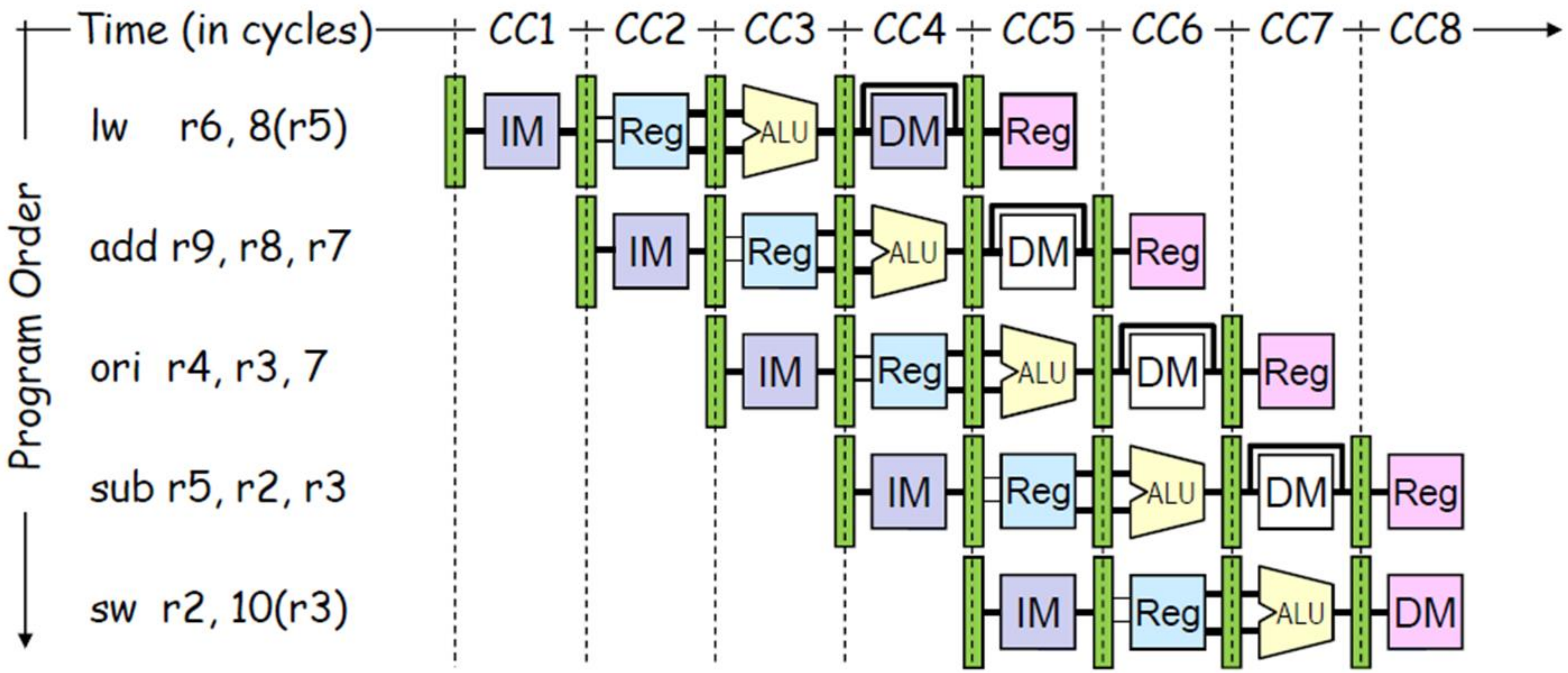
6. Performance Through Parallelism

Doing different parts of a task in parallel accomplishes the task in less time than doing them sequentially. A processor engages in several activities in the execution of an instruction. It runs faster if it can do these activities in parallel.

- Parallel Requests
  Assigned to computer, e.g., Search “Katz”
- Parallel Threads,
  Assigned to core, e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time, GPU is SIMD (Single Instruction Multiple Word)
- Hardware Descriptions
  All gates functioning in parallel at same time
- Programming Languages

Performance Through Pipelining

This idea is an extension of the idea of parallelism. It is essentially handling the activities involved in instruction execution as an assembly line. As soon as the first activity of an instruction is done you move it to the second activity and start the first activity of a new instruction. This results in executing more instructions per unit time compared to waiting for all activities of the first instruction to complete before starting the second instruction.

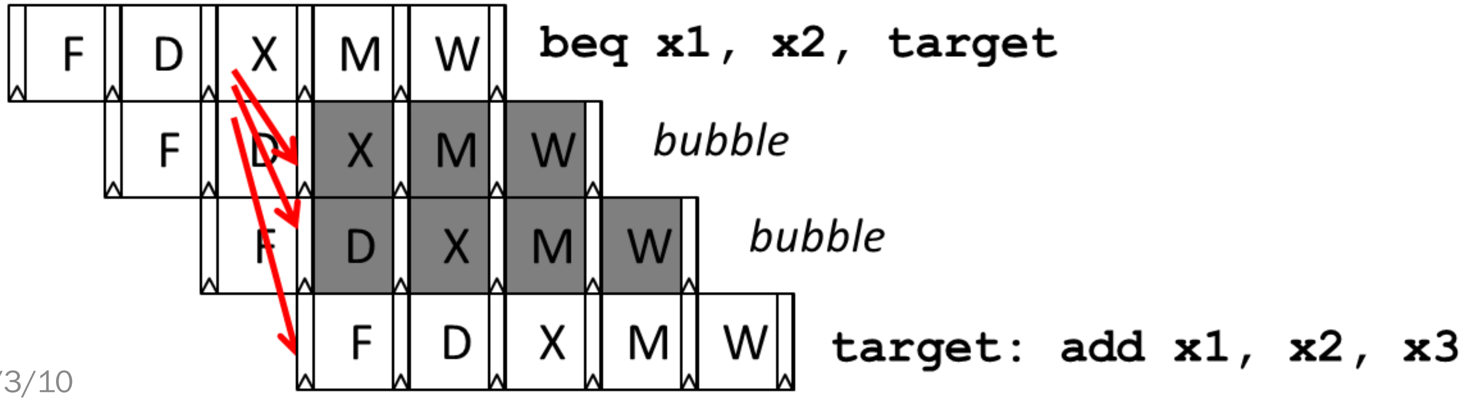


8. Performance Through Prediction

A conditional branch is a type of instruction determines the next instruction to be executed based on a condition test. Conditional branches are essential for implementing high-level language if statements and loops.

Unfortunately, conditional branches interfere with the smooth operation of a pipeline — the processor does not know where to fetch the next instruction until after the condition has been tested.

Many modern processors reduce the impact of branches with speculative execution: make an informed guess about the outcome of the condition test and start executing the indicated instruction. Performance is improved if the guesses are reasonably accurate and the penalty of wrong guesses is not too severe.


F D X M W beq x1, x2, target
F D X M W bubble
F D X M W bubble
F D X M W target: add x1, x2, x3

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - John Hennessy (Standford)and David Patterson (UCB)
  - Chenxi Zhang (Tongji)
  - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**