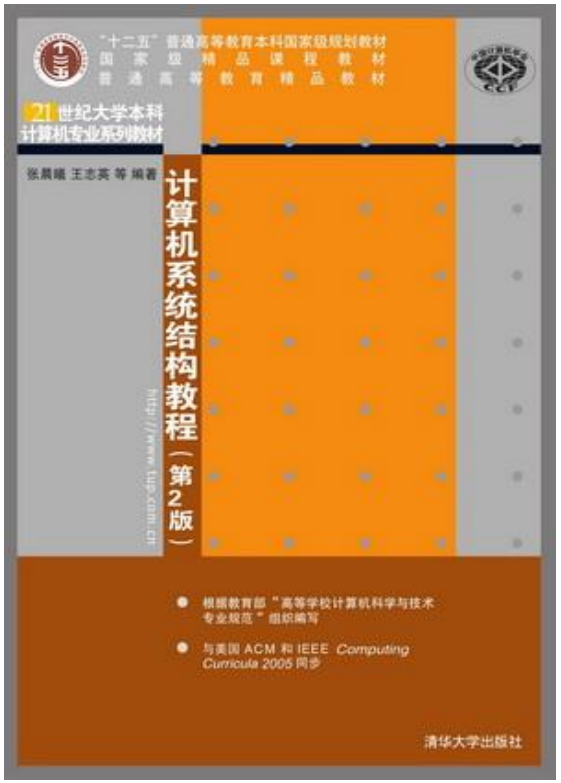
计算机系统结构

Computer Architecture



主讲人：周可

华中科技大学教育部信息存储系统重点实验室

武汉光电国家研究中心

zhke@hust.edu.cn

https://idsm.wnlo.hust.edu.cn/

参考书籍

美国计算机协会（ACM）3月21日宣布，
前斯坦福大学校长John L. Hennessy
和加州大学伯克利分校退休教授David A. Patterson
获得2017 ACM 图灵奖，以表彰他们在计算机体系结构的设计和评估方面开创了一套系统的、量化的方法，并对微处理器行业产生了深远的影响。

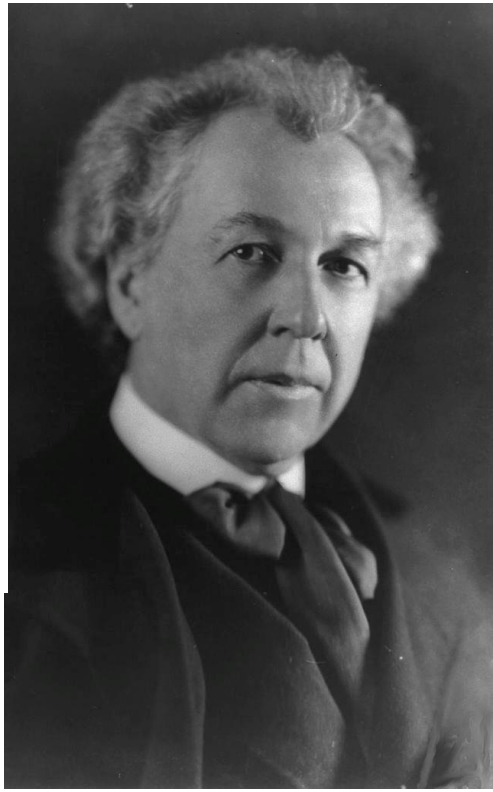
从建筑（Architecture）开始

Fallingwater

From Wikipedia, the free encyclopedia

Fallingwater or **Kaufmann Residence** is a house designed by architect Frank Lloyd Wright in 1935 in rural southwestern Pennsylvania, 43 miles (69 km) southeast of Pittsburgh.[4] The home was built partly over a waterfall on Bear Run in the Mill Run section of Stewart Township, Fayette County, Pennsylvania, in the Laurel Highlands of the Allegheny Mountains.

Time cited it after its completion as Wright's "most beautiful job";[5] it is listed among *Smithsonian*'s Life List of 28 places "to visit before you die."[6] It was designated a National Historic Landmark in 1966.[3] In 1991, members of the American Institute of Architects named the house the "best all-time work of American architecture" and in 2007, it was ranked twenty-ninth on the list of America's Favorite Architecture according to the AIA.



Fallingwater，流水别墅/流水居，坐落于美国宾夕法尼亚州匹兹堡东南方的森林里，于1934年由美国建筑师Frand Lloyd Wright设计。别墅以建在溪流正上方的独特设计而广为人知，是著名的现代建筑，被美国建筑协会誉为“美国建筑史上最伟大的作品”。

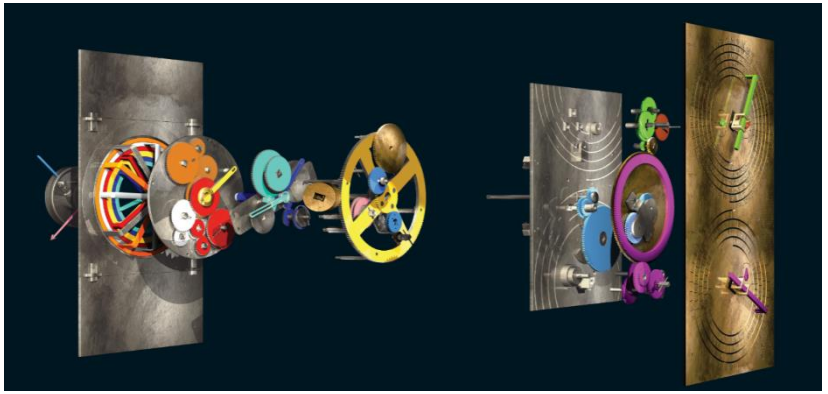
Many guesses

- (Ultra) hard work, perseverance, dedication (over decades)
- Experience of decades
- Creativity
- Out-of-the-box thinking（不落窠臼）
- Principled design
- A good understanding of past designs
- Good judgment and intuition
- Strong combination of skills (math, architecture, art, …)
- …

A Quote from The Architect Himself

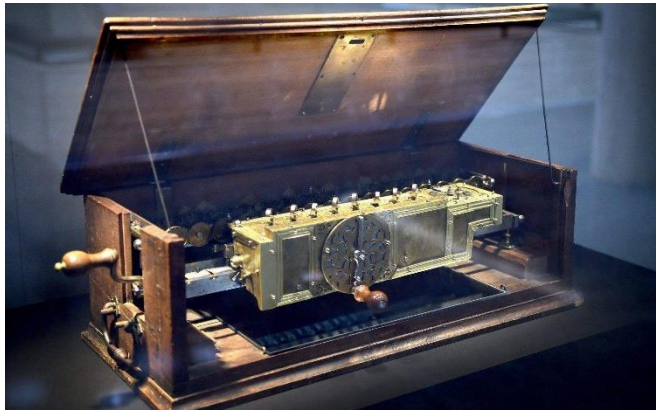
"architecture [...] based upon **principle**, and not upon **precedent**"

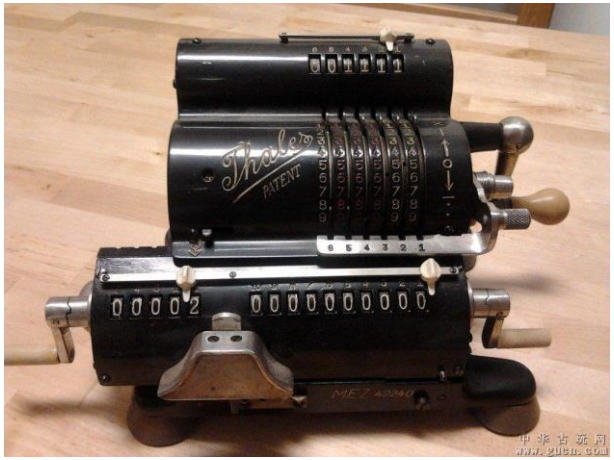
安提基特拉机械



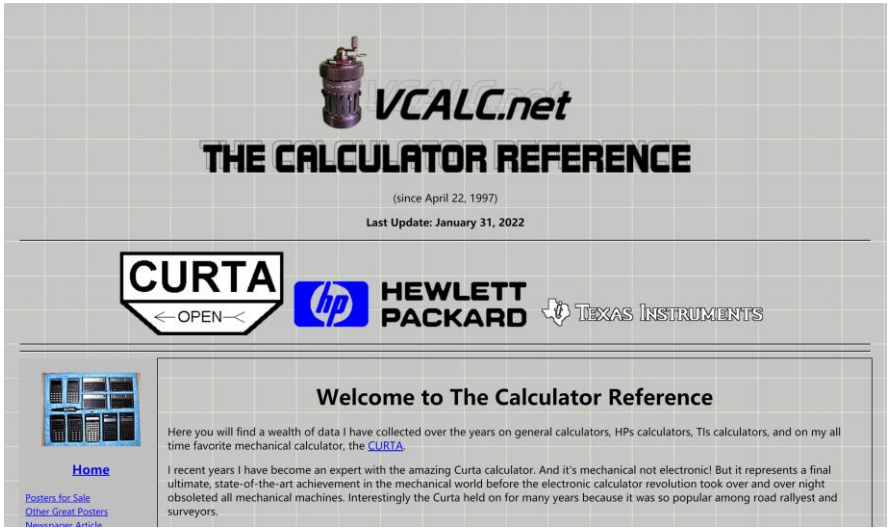
机械计算器

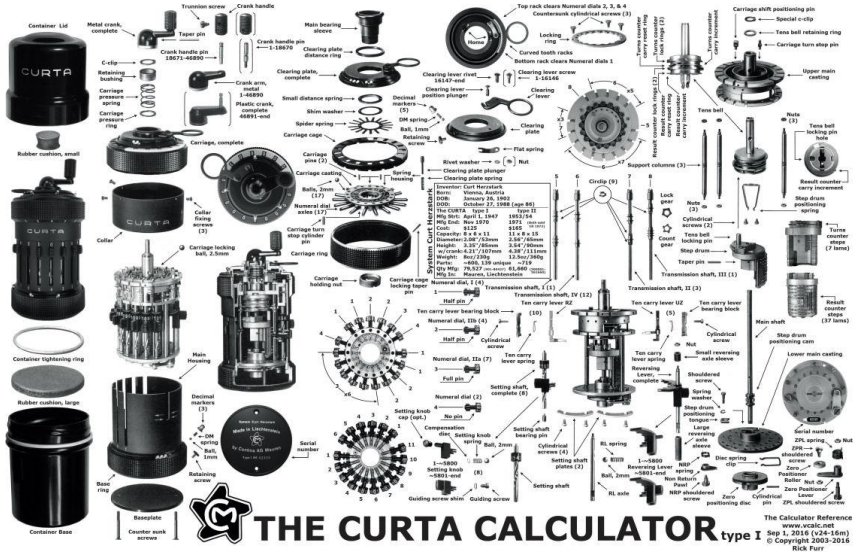
莱布尼茨（G.W. Leibniz1646-1716）
德国数学家　　二进制





www.vcalc.net





B站

【中文字幕】Curta机械计算器的工作原理

3.7万 78 2016-03-05 06:23:09

Major High-Level Goals of This Course

1. Understand the principles
2. Understand the precedents
3. Based on such understanding:
   - Enable you to evaluate tradeoffs of different designs and ideas
   - Enable you to develop principled designs
   - Enable you to develop novel, out-of-the-box designs
4. The focus is on:
   - Principles, precedents, and how to use them for new designs

In Computer Architecture

1. Two key goals of this course are

   - to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer

   - to enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components

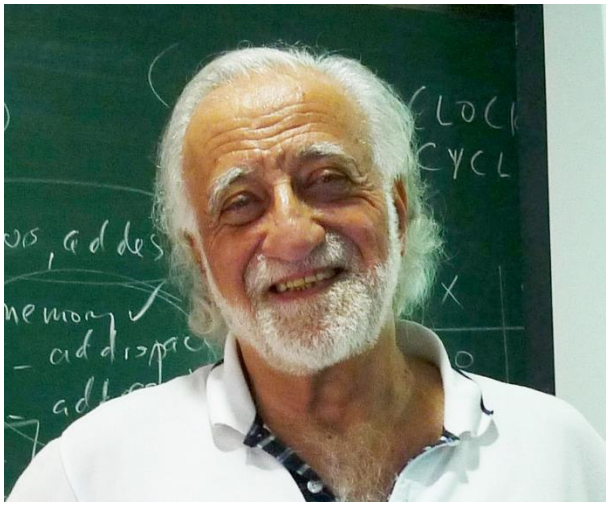
Computer Architecture’ s Changing Definition

1. 1950s to 1960s: Computer Architecture Course: Computer Arithmetic
2. 1970s to mid 1980s: Computer Architecture Course: Instruction Set Design, especially ISA appropriate for compilers
3. 1990s: Computer Architecture Course: Design of CPU, memory system, I/O system, Multiprocessors, Networks
4. 2000s: Multi-core design, on-chip networking, parallel programming paradigms, power reduction
5. 2010s: Computer Architecture Course: Self adapting systems? Self organizing structures? DNA Systems/Quantum Computing?

Role of the (Computer) Architect

Role of the Architect

- *-- Look Backward (Examine old code)*
- *-- Look forward (Listen to the dreamers)*
- *-- Look Up (Nature of the problems)*
- *-- Look Down (Predict the future of technology)*



IEEE Spectrum评为美国计算机界的卓越泰斗（与《计算机程序设计艺术》的作者，图灵奖获得者Donald Knuth齐名，全球只有他们俩人享此殊荣），在美国乃至世界计算机体系结构领域有着广泛的影响力。

from Yale Patt's lecture notes

1. Look backward (to the past)
   - Understand tradeoffs and designs, upsides/downsides, past workloads. Analyze and evaluate the past.
2. Look forward (to the future)
   - Be the dreamer and create new designs. Listen to dreamers.
   - Push the state of the art. Evaluate new design choices.
3. Look up (towards problems in the computing stack)
   - Understand important problems and their nature.
   - Develop architectures and ideas to solve important problems.
4. Look down (towards device/circuit technology)
   - Understand the capabilities of the underlying technology.
   - Predict and adapt to the future of technology (you are designing for N years ahead). Enable the future technology.

第1章 计算机系统结构的基础知识

1.1 计算机系统结构的基本概念

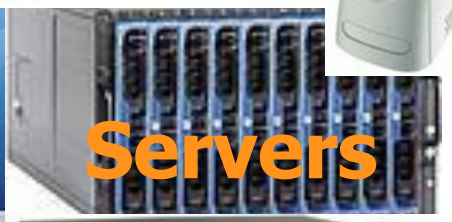
1.2 计算机系统的设计

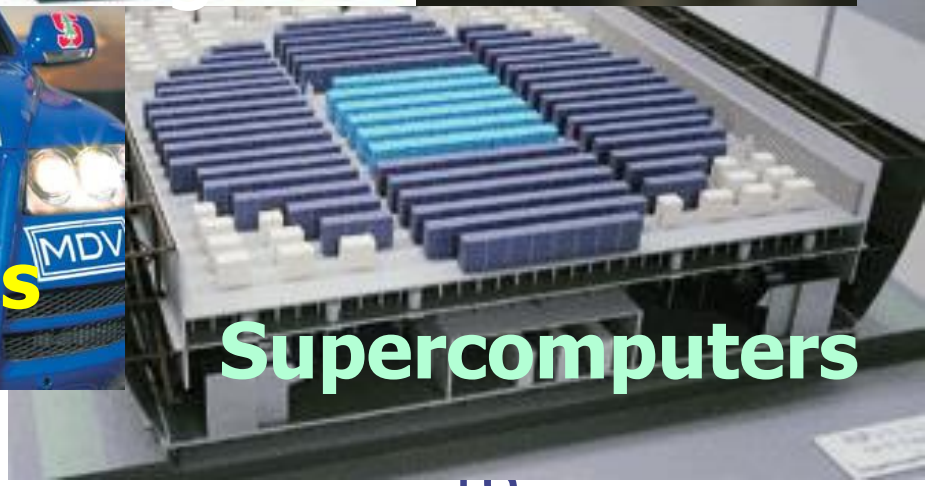
1.3 计算机系统的性能评测

1.4 计算机系统结构的发展

1.5 计算机系统结构中并行性的发展

Computing Devices Now



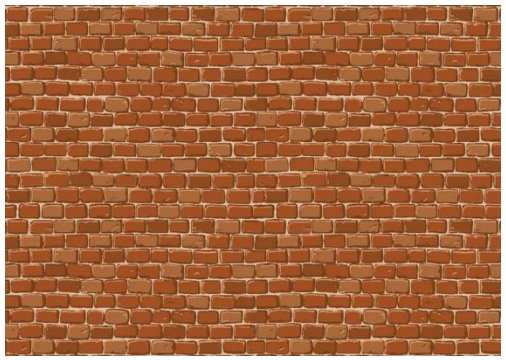


1.1 计算机系统结构的基本概念

1. 第一台通用电子计算机诞生于1946年
2. 计算机技术的飞速发展受益于两个方面
   - 计算机制造技术的发展
   - 计算机系统结构的创新
3. 经历了四个发展过程

时　间	原　因	每年的性能增长
1946年起的25年	两种因素都起着主要的作用	25%
20世纪70年代末～80年代初	大规模集成电路和微处理器出现，以集成电路为代表的制造技术的发展	约35%
80年代中开始	RISC结构的出现，系统结构不断更新和变革，制造技术不断发展	50%以上 维持了约16年
2002年以来	3个（见下页）	约22%

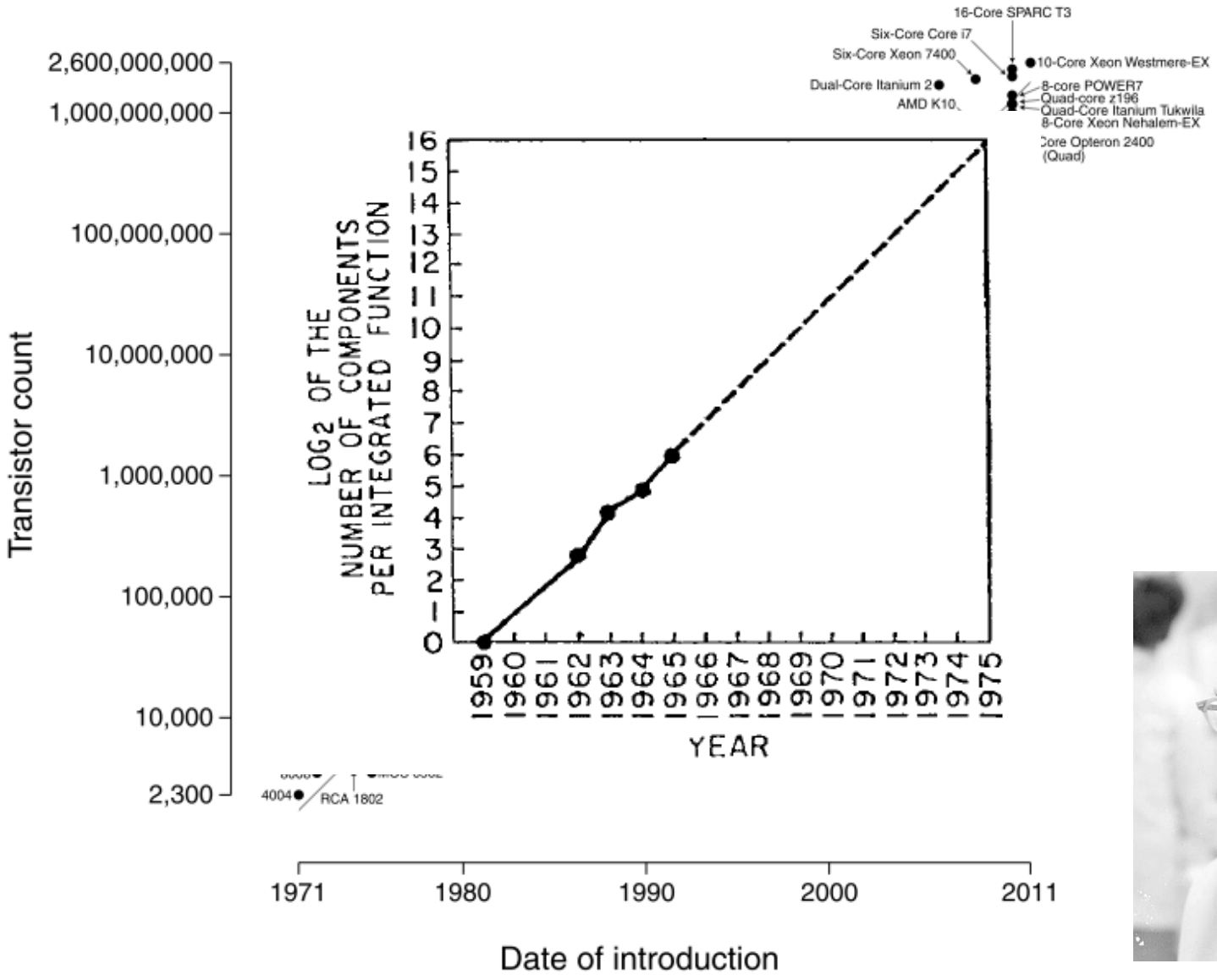
- 可以进一步有效地开发的指令级并行性已经很少
- 功耗大问题 （Power Wall）
- 存储器访问速度的提高缓慢 （Memory, I/O Wall）


2004年发生了什么？系统结构的重大转折：

从单纯依靠指令级并行转向开发线程级并行和数据级并行

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Number of transistors on an integrated circuit doubles ~ every two years

Image source: Wikipedia

- 计算机系统结构是什么？

广义定义：程序员必须掌握的计算机属性（P3，Amdahl定义）

狭义定义：机器语言程序员必须掌握的计算机属性
（包括：数据表示、寻址规则、寄存器组织、指令系统、中断机构、机器工作状态、存储系统、信息保护手段、输入/输出结构与管理）

理解：如果两台计算机的系统结构相同，则它们对软件来说具有完全相同的属性，所以软件在它们上面是互相通用的，即互为“兼容机”。

兼容机：指系统结构相同而组成、实现技术不同的计算机。

- 计算机系统结构这门课教什么？

学科定义：通过硬软件结合方法提高计算机性能的理论与技术

说明：

1. 计算机性能指速度、容量等；

2. 提高计算机性能的硬件技术———由“组成原理”负责（如并行主存）；

3. 提高计算机性能的软件技术———由“计算方法”负责（指各种优化算法）。

举例：

“虚拟存储器”就是一种通过硬软件结合方法提高计算机存储容量的技术。

1.2 计算机系统的设计

1.2.1 计算机系统设计的定量原理

4个定量原理：

1. 以经常性事件为重点

- 对经常发生的情况采用优化方法的原则进行选择，以得到更多的总体上的改进
- 优化是指分配更多的资源、达到更高的性能或者分配更多的电能等

Make the common case faster | Amdahl

2. Amdahl定律

加快某部件执行速度所能获得的系统性能加速比，受限于该部件的执行时间占系统中总执行时间的百分比。

系统性能加速比：

$$\text{加速比} = \frac{\text{系统性能}_{\text{改进后}}}{\text{系统性能}_{\text{改进前}}} = \frac{\text{总执行时间}_{\text{改进前}}}{\text{总执行时间}_{\text{改进后}}}$$

➢ 加速比依赖于两个因素

- 可改进比例（F_e）：在改进前的系统中，可改进部分的执行时间在总的执行时间中所占的比例。

  它总是小于等于1。

例如：一个需运行60秒的程序中有20秒的运算可以加速，

那么这个比例就是20/60。

- 部件加速比（S_e）：可改进部分改进以后性能提高的倍数。它是改进前所需的执行时间与改进后执行时间的比。

  一般情况下部件加速比是大于1的。

例如：若系统改进后，可改进部分的执行时间是2秒，

而改进前其执行时间为5秒，则部件加速比为5/2。

- 改进后程序的总执行时间T_n

$$T_n = T_0\left(1 - Fe + \frac{Fe}{Se}\right)$$

  - T_0：改进前整个程序的执行时间
  - $1-F_e$：不可改进比例

系统加速比S_n为改进前与改进后总执行时间之比:

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - Fe) + \frac{Fe}{Se}}$$

例1.1 将计算机系统中某一功能的处理速度加快15倍，但该功能的处理时间仅占整个系统运行时间的40%，则采用此增强功能方法后，能使整个系统的性能提高多少？

解 由题可知： $F_e = 40\% = 0.4$

$S_e = 15$

根据Amdahl定律可知：

$$S_n = \frac{1}{(1-Fe)+\frac{Fe}{Se}} = \frac{1}{(1-0.4)+\frac{0.4}{15}} \approx 1.6$$

采用此增强功能方法后，能使整个系统的性能提高到原来的1.6倍。

- Amdahl定律：如果仅仅对计算任务中的一部分做性能改进，则改进得越多，所得到的总体性能的提升就越有限。
- 重要推论：如果只针对整个任务的一部分进行改进和优化，那么所获得的加速比不超过：

1/（1－可改进比例）

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1-Fe)+\frac{Fe}{Se}} \qquad S_e \to \infty$$

3. CPU性能公式

- 执行一个程序所需的CPU时间

CPU时间 = 执行程序所需的时钟周期数×时钟周期时间

其中：时钟周期时间是系统时钟频率的倒数。

- 每条指令执行的平均时钟周期数CPI

（Cycles Per Instruction）

CPI = 执行程序所需的时钟周期数 / IC

IC：所执行的指令条数

- 程序执行的CPU时间可以写成

CPU时间 = IC ×CPI ×时钟周期时间

- CPU的性能取决于三个参数
  - 时钟周期时间：取决于硬件实现技术和计算机组成；
  - CPI：取决于计算机组成和指令系统的结构；
  - IC：取决于指令系统的结构和编译技术。
- 对CPU性能公式进行进一步细化

假设：计算机系统有n种指令；

CPI_i ：第i种指令的处理时间；

IC_i ：在程序中第i种指令出现的次数；

则：

$$\text{CPU时钟周期数} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

CPU时间 = 执行程序所需的时钟周期数×时钟周期时间

$$= \sum_{i=1}^{n} (CPI_i \times IC_i) \times 时钟周期时间$$

CPI可以表示为：

$$CPI = \frac{时钟周期数}{IC} = \frac{\sum_{i=1}^{n} (CPI_i \times IC_i)}{IC} = \sum_{i=1}^{n} (CPI_i \times \frac{IC_i}{IC})$$

其中：(IC_i/IC)反映了第i种指令在程序中所占的比例。

例1.3 假设FP指令的比例为25%，其中，FPSQR占全部指令的比例为2%，FP操作的CPI为4，FPSQR操作的CPI为20，其他指令的平均CPI为1.33。现有两种改进方案，第一种是把FPSQR操作的CPI减至2，第二种是把所有的FP操作的CPI减至2，试比较两种方案对系统性能的提高程度。

解 没有改进之前，每条指令的平均时钟周期CPI为：

$$CPI = \sum_{i=1}^{n}\left(CPI_i \times \frac{IC_i}{IC}\right) = (4 \times 25\%) + (1.33 \times 75\%) = 2$$

（1）采用第一种方案

FPSQR操作的CPI由$CPI_{FPSQR}=20$减至$CPI'_{FPSQR}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_1=CPI—（CPI_{FPSQR}—CPI'_{FPSQR}）\times 2\%$$
$$=2—(20—2)\times 2\%=1.64$$

（2）采用第二种方案

所有FP操作的CPI由$CPI_{FP}=4$减至$CPI'_{FP}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_2=CPI—（CPI_{FP}—CPI'_{FP}）\times 25\%$$
$$=2—(4—2)\times 25\%=1.5$$

从降低整个系统的指令平均时钟周期数的程度来看，第二种方案优于第一种方案。

4. 程序的局部性原理 (Locality)

程序执行时所访问的存储器地址分布不是随机的，而是相对地簇聚。

- 常用的一个经验规则

  程序执行时间的90%都是在执行程序中10%的代码。

- 程序的时间局部性 (Temporal Locality)

  程序即将用到的信息很可能就是目前正在使用的信息。

- 程序的空间局部性 (Spatial Locality)

  程序即将用到的信息很可能与目前正在使用的信息在空间上相邻或者临近

1.3 计算机系统的性能评测

1. 执行时间和吞吐率

如何评测一台计算机的性能，与测试者看问题的角度有关。

- 用户关心的是：单个程序的执行时间（执行单个程序所花的时间很少）
- 数据处理中心的管理员关心的是：吞吐率（在单位时间里能够完成的任务很多 ）

假设两台计算机为X和Y，X比Y快的意思是：

对于给定任务，X的执行时间比Y的执行时间少。

X的性能是Y的n倍：

$$\frac{\text{执行时间Y}}{\text{执行时间X}}=n$$

执行时间与性能成反比：

$$n=\frac{\text{执行时间Y}}{\text{执行时间X}}=\frac{\frac{1}{\text{性能Y}}}{\frac{1}{\text{性能X}}}=\frac{\text{性能X}}{\text{性能Y}}$$

- 执行时间可以有多种定义：
  - 计算机完成某一任务所花费的全部时间，包括磁盘访问、存储器访问、输入/输出、操作系统开销等。
  - **CPU时间：**CPU执行所给定的程序所花费的时间，不包含I/O等待时间以及运行其它程序的时间。
    - 用户CPU时间：用户程序所耗费的CPU时间。
    - 系统CPU时间：用户程序运行期间操作系统耗费的CPU时间。

2. 基准测试程序

- 用于测试和比较性能的基准测试程序的最佳选择是真实应用程序。

  （例如编译器）

- 以前常采用简化了的程序，例如：
  - 核心测试程序：从真实程序中选出的关键代码段构成的小程序。
  - 小测试程序：简单的只有几十行的小程序。
  - 合成的测试程序：人工合成出来的程序。

从测试性能的角度来看，上述测试程序不可信。

原因：

- 这些程序比较小，具有片面性；
- 系统结构设计者和编译器的设计者可以“合谋”把他们的机器面向这些测试程序进行优化设计，使得该机器显得性能更高。

➢ 性能测试的结果除了和采用什么测试程序有关以外，还和在什么条件下进行测试有关。

➢ 基准测试程序设计者对制造商的要求

- 采用同一种编译器；
- 对同一种语言的程序都采用相同的一组编译标志。

- 基准测试程序套件：由各种不同的真实应用程序构成。

  （能比较全面地反映计算机在各个方面的处理性能）

- SPEC系列：最成功和最常见的测试程序套件

  （美国的标准性能测试公司创建）

SPEC CPU2006:

整数程序12个（CINT2006）

9个是用C写的，3个是用C++写的

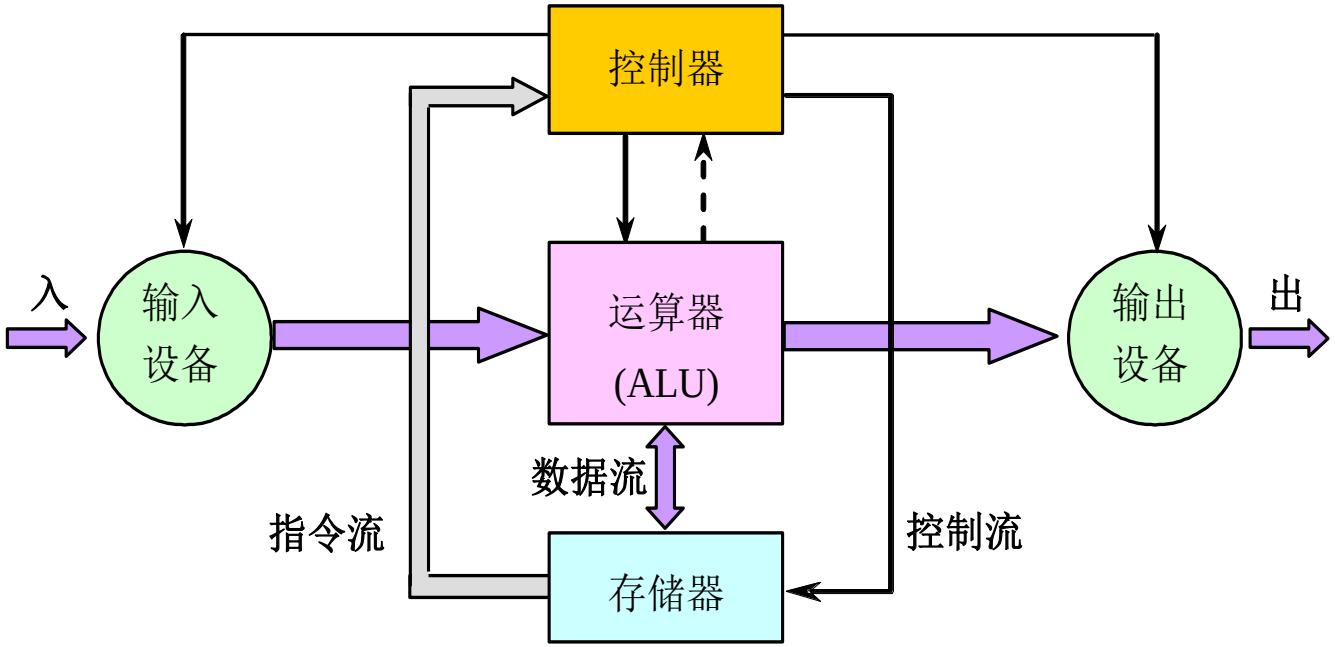
浮点程序17个（CFP2006）

6个是用FORTRAN写的，4个是用C++写的，3个是用C写的，4个是用C和FORTRAN混合编写的。

- SPEC测试程序套件中的其它一系列测试程序组件
  - SPECSFS：用于NFS（网络文件系统）文件服务器的测试程序。它不仅测试处理器的性能，而且测试I/O系统的性能。它重点测试吞吐率。
  - SPECWeb：Web服务器测试程序。
  - SPECviewperf：用于测试图形系统支持OpenGL库的性能。
  - SPECapc：用于测试图形密集型应用的性能

1.4 计算机系统结构的发展

1.4.1 冯·诺依曼结构及其改进



存储程序计算机的结构

1. 存储程序原理的基本点：指令驱动

程序预先存放在计算机存储器中，机器一旦启动，就能按照程序指定的逻辑顺序执行这些程序，自动完成由程序所描述的处理工作。

2. 冯·诺依曼结构的主要特点

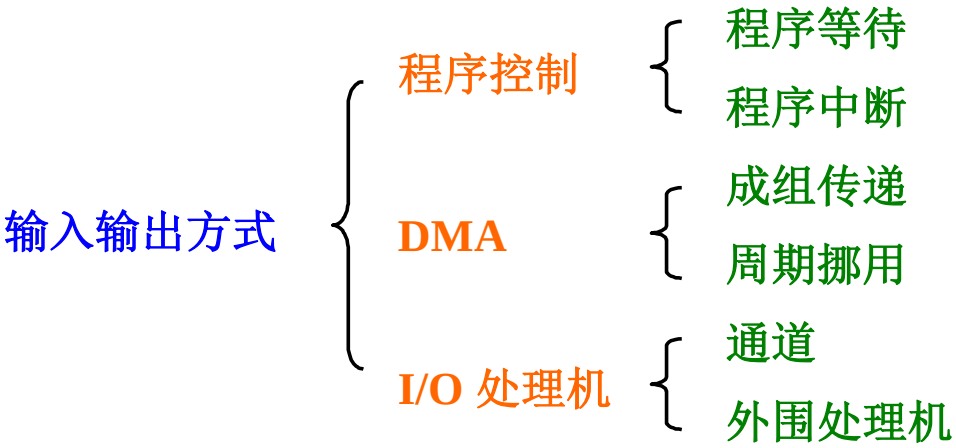
- 计算机以运算器为中心。
- 在存储器中，指令和数据同等对待。

  指令和数据一样可以进行运算，即由指令组成的程序是可以修改的。
- 存储器是按地址访问、按顺序线性编址的一维结构，每个单元的位数是固定的。

- 指令的执行是顺序的。
  - 一般是按照指令在存储器中存放的顺序执行。
  - 程序的分支由转移指令实现。
  - 由指令计数器PC指明当前正在执行的指令在存储器中的地址。
- 指令由操作码和地址码组成。
- 指令和数据均以二进制编码表示，采用二进制运算。

3. 对系统结构进行的改进

➢ 输入/输出方式的改进



➢ 采用并行处理技术

- 如何挖掘传统机器中的并行性？
- 在不同的级别采用并行技术。

  例如：微操作级、指令级、线程级、进程级、任务级等。

- 存储器组织结构的发展
  - 相联存储器与相联处理机
  - 通用寄存器组
  - 高速缓冲存储器Cache
- 指令系统的发展

  两个发展方向：
  - 复杂指令集计算机CISC
  - 精减指令集计算机RISC

1.4.3 器件发展对系统结构的影响

1. 推动计算机系统结构不断发展的最活跃的因素
2. 摩尔定律

   集成电路芯片上所集成的晶体管数目每隔18个月就翻一番。
3. 计算机的分代主要以器件作为划分标准。
   - 它们在器件、系统结构和软件技术等方面都有各自的特征。
     - **SMP**：对称式共享存储器多处理机

       MPP：大规模并行处理机　　**MP**：多处理机

分代	器件特征	结构特征	软件特征	典型实例
第一代 （1945—1954年）	电子管和继电器	存储程序计算机 程序控制I/O	机器语言 汇编语言	普林斯顿ISA， ENIAC，IBM 701
第二代 （1955—1964年）	晶体管、磁芯 印刷电路	浮点数据表示 寻址技术 中断、I/O处理机	高级语言和编译 批处理监控系统	Univac LAPC， CDC 1604， IBM 7030
第三代 （1965—1974年）	SSI和MSI 多层印刷电路 微程序	流水线、Cache 先行处理 系列机	多道程序 分时操作系统	IBM 360/370， CDC 6600/7600， DEC PDP-8
第四代 （1975—1990年）	LSI和VLSI 半导体存储器	向量处理 分布式存储器	并行与分布处理	Cray-1，IBM 3090，DEC VAX 9000，Convax-1
第五代 （1991年—）	高性能微处理器 高密度电路	超标量、超流水 SMP、MP、MPP 机群	大规模、可扩展 并行与分布处理	SGI Cray T3E， IBM SP2，DEC AlphaServer 8400

1.4.4 应用对系统结构的影响

1. 不同的应用对计算机系统结构的设计提出了不同的要求。
2. 应用需求是促使计算机系统结构发展的最根本的动力。
3. 一些特殊领域：需要高性能的系统结构
   - 高结构化的数值计算

     气象模型、流体动力学、有限元分析
   - 非结构化的数值计算

     蒙特卡洛模拟、稀疏矩阵
   - 实时多因素问题

     语音识别、图象处理、计算机视觉

- 大存储容量和输入输出密集的问题

  数据库系统、事务处理系统

- 图形学和设计问题

  计算机辅助设计

- 人工智能

  面向知识的系统、推理系统等

Big Data:

Volume, Velocity, Variety, Veracity, Value

1.5 计算机系统结构中并行性的发展

1.5.1 并行性的概念

1. **并行性**：计算机系统在同一时刻或者同一时间间隔内进行多种运算或操作。

只要在时间上相互重叠，就存在并行性。

- **同时性**：两个或两个以上的事件在同一时刻发生。
- **并发性**：两个或两个以上的事件在同一时间间隔内发生。

2. 从处理数据的角度来看，并行性等级从低到高可分为：

- 字串位串：每次只对一个字的一位进行处理。

  最基本的串行处理方式，不存在并行性。

- 字串位并：同时对一个字的全部位进行处理，不同字之间是串行的。

  开始出现并行性。

- 字并位串：同时对许多字的同一位（称为位片）进行处理。

  具有较高的并行性。

- 全并行：同时对许多字的全部位或部分位进行处理。

  最高一级的并行。

3. 从执行程序的角度来看，并行性等级从低到高可分为：

- 指令内部并行：单条指令中各微操作之间的并行。
- 指令级并行：并行执行两条或两条以上的指令。
- 线程级并行：并行执行两个或两个以上的线程。

  通常是以一个进程内派生的多个线程为调度单位。
- 任务级或过程级并行：并行执行两个或两个以上的过程或任务（程序段）

  以子程序或进程为调度单元。
- 作业或程序级并行：并行执行两个或两个以上的作业或程序。

1.5.2 提高并行性的技术途径：

1. 时间重叠

引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

2. 资源重复

引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能。

3. 资源共享

这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。

1.5.3 单机系统中并行性的发展

1. 在发展高性能单处理机过程中，起主导作用的是时间重叠原理。

实现时间重叠的基础：部件功能专用化

- 把一件工作按功能分割为若干相互联系的部分；
- 把每一部分指定给专门的部件完成；
- 然后按时间重叠原理把各部分的执行过程在时间上重叠起来，使所有部件依次分工完成一组同样的工作。

2. 在单处理机中，资源重复原理的运用也已经十分普遍。

- 多体存储器
- 多操作部件
  - 通用部件被分解成若干个专用部件，如加法部件、乘法部件、除法部件、逻辑运算部件等，而且同一种部件也可以重复设置多个。
  - 只要指令所需的操作部件空闲，就可以开始执行这条指令（如果操作数已准备好的话）。
  - 这实现了指令级并行。

- 阵列处理机（并行处理机）

  更进一步，设置许多相同的处理单元，让它们在同一个控制器的指挥下，按照同一条指令的要求，对向量或数组的各元素同时进行同一操作，就形成了阵列处理机 。

3. 在单处理机中，资源共享的概念实质上是用单处理机模拟多处理机的功能，形成所谓虚拟机的概念。
   - 分时系统

1.5.4 多机系统中并行性的发展

1. 多机系统遵循时间重叠、资源重复、资源共享原理，发展为3种不同的多处理机：

   同构型多处理机、异构型多处理机、分布式系统

2. 耦合度

   反映多机系统中各机器之间物理连接的紧密程度和交互作用能力的强弱。

   - 紧密耦合系统（直接耦合系统）：在这种系统中，计算机之间的物理连接的频带较高，一般是

通过总线或高速开关互连，可以共享主存。

- 松散耦合系统（间接耦合系统）：一般是通过通道或通信线路实现计算机之间的互连，可以共享外存设备（磁盘、磁带等）。机器之间的相互作用是在文件或数据集一级上进行。

  表现为两种形式：

  - 多台计算机和共享的外存设备连接，不同机器之间实现功能上的分工（功能专用化），机器处理的结果以文件或数据集的形式送到共享外存设备，供其它机器继续处理。
  - 计算机网，通过通信线路连接，实现更大范围的资源共享。

3. 功能专用化（实现时间重叠）

- 专用外围处理机

  例如：输入/输出功能的分离

- 专用处理机

  如数组运算、高级语言翻译、数据库管理等，分离出来。

- 异构型多处理机系统

  由多个不同类型、至少担负不同功能的处理机组成，它们按照作业要求的顺序，利用时间重叠原理，依次对它们的多个任务进行加工，各自完成规定的功能动作。

4. 机间互连

- 容错系统
- 可重构系统

对计算机之间互连网络的性能提出了更高的要求。高带宽、低延迟、低开销的机间互连网络是高效实现程序或任务一级并行处理的前提条件。

- 同构型多处理机系统

由多个同类型或至少担负同等功能的处理机组成，它们同时处理同一作业中能并行执行的多个任务。

(1)计算机系统的多级层次模型，计算机系统结构的广义定义与狭义定义，与组织、实现的关系；
(2)计算机系统结构的分类；
(3)计算机系统设计的定量原理
Amdahl定律
CPU性能公式
平均时钟周期数CPI
每秒百万指令数MIPS
(4)执行时间、吞吐率和基准测试程序；
(5)冯.诺依曼结构的特点；
(6)并行性的等级与技术途径。

作业：1.7，1.10，1.11,下周二上课的时候交作业

其它要求

- 考试成绩60%+实验20%+平时作业20%
- 作业不用抄题，写明题号
- 作业本要写明班级学号，便于记录
- 实验要求独立完成，认真撰写实验报告