第一章 计算机系统结构的基础知识

2020年5月8日 15:08

1.1 计算机系统结构的基本概念

1.1.1计算机系统的层次结构

- 第0级：硬联逻辑
- 由硬件实现

- 第1级：微程序
- 控制信息在各程序之间传递，由微程序固件实现
- 0-1 属于硬件，实现机器特定功能的中央控制部分

- 第2级：机器语言机器 传统机器语言机器
- 该机的指令系统，解释工作由第1级的微程序控制，是软硬件的分界

- 第3级：操作系统机器
- 包含传统机器语言的多数指令（算术运算、逻辑运算etc），还包括操作系统级指令（文件打开关闭读写etc）。是运行在第2级机器上的解释程序

- 第4级：汇编语言机器
- 汇编程序：将汇编语言翻译成第2、3级语言的程序

- 第5级：高级语言机器
- 编译程序将高级语言翻译成第3、4级语言

- 第6级：应用程序

- 翻译：将高一级的语言通过编译器转化为低一级的语言
- 固件：具有软件功能的硬件
- 虚拟机：两种实现方式：（1）纯软件（2）硬件+固件

- 物理机与虚拟机的区别：
- 虚拟机没有硬件实体，而物理机有实体。也就是说，物理机是有实体的硬件系统，比如服务器等，而虚拟机是借助物理机虚拟出虚拟的硬件系统。
- 两者容器不同。以电脑举例，一台电脑就是一个物理机，但是一台电脑可以成为多个虚拟机，每个虚拟机可运行不同的操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响。
- 物理机的执行引擎是直接建立在CPU处理器、指令集、操作系统和硬件层面上的；而虚拟机的执行引擎则由自己实现，因此可以制定自己的指令集和执行引擎的结构体系，而且还可以执行一些不被硬件直接支持的指令集格式。

1.1.2计算机系统结构定义

- 计算机系统结构定义一
  系统结构：程序设计者所看到的一个计算机系统的属性，即概念性结构和功能特性，由Amdahl提出。
  功能特性：

- 数据表示：硬件能够直接认别和处理的数据结构
- 寻址技术：编址方式、寻址方式和定位方式
- 寄存器组织：操作数寄存器、变址寄存器、控制寄存器和专用寄存器的定义、数量和使用规则。
- 指令系统：操作类型、格式，指令间的排序控制
- 中断系统：中断类型、中断级别和中断响应方式
- 存储系统：寻址空间、虚拟存储器、Cache存储器
- 处理机工作状态：定义和切换方式
- 输入输出系统：数据交换方式、交换过程的控制
- 信息保护：信息保护方式和硬件对信息保护的支持
- 计算机系统结构定义二
- 系统结构：计算机系统由软件、固件和硬件组成，它们在功能上是同等的，同一种功能可以用硬件实现，也可以用软件和固件实现。
- **计算机系统结构定义（J.L.Hennessy和D.A.Patterson):**
- **指令系统，组成，硬件。**

1.1.3 计算机组成与实现

1.计算机组成

- 计算机组成是指计算机系统结构的逻辑实现。

主要包括：

（1）确定数据通路的宽度；
（2）确定各种操作对功能部件的共享程度；
（3）确定专用的功能部件；
（4）确定功能部件的并行度；
（5）设计缓冲和排队策略；
（6）设计控制机构；
（7）确定采用何种可靠性技术。

2.计算机实现

- 计算机实现是指计算机组成的物理实现。

主要包括：

（1）处理机、主存储器等部件的物理结构；
（2）器件的集成度和速度；
（3）专用器件的设计；
（4）器件、模块、插件、底版的划分与连接；
（5）信号传输技术；
（6）电源、冷却及装配技术，制造工艺及技术。

3.系统结构、组成和实现三者的关系

- 计算机组成是计算机系统结构的逻辑实现，计算机实现是计算机组成的物理实现，三者各自包含不同的内容，但又有着紧密的联系。
- 一种系统结构可以有多种组成，同样，一种组成可以有多种物理实现。

1.1.4 计算机系统结构的分类

- 1.1.4.1弗林(Flynn)分类法

弗林(Flynn)分类法按照指令流(Instruction stream,IS) 和数据流(Data stream, DS)的多倍性进行分类。

指令流：计算机执行的指令序列。

数据流：由指令流调用的数据系列。

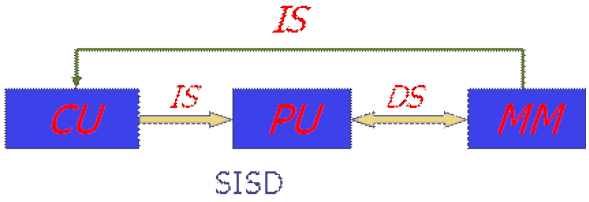
多倍性：在系统最受限的部件上，同时处于同一执行阶段的指令或数据的最大数目。

弗林(Flynn)分类法把计算机系统结构分为以下四类：

1.单指令流单数据流SISD(Single IS Single DS)

IS：指令流　DS：数据流　CU：控制器　PU：处理器

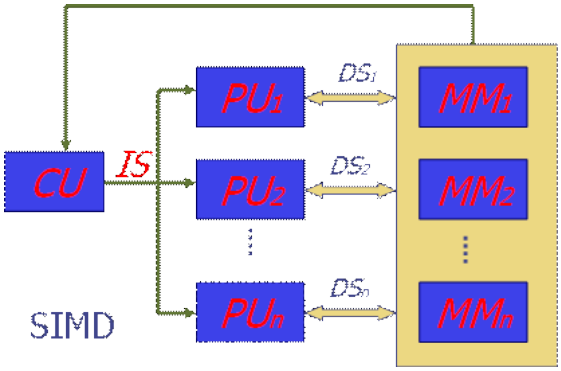
MM：存储器



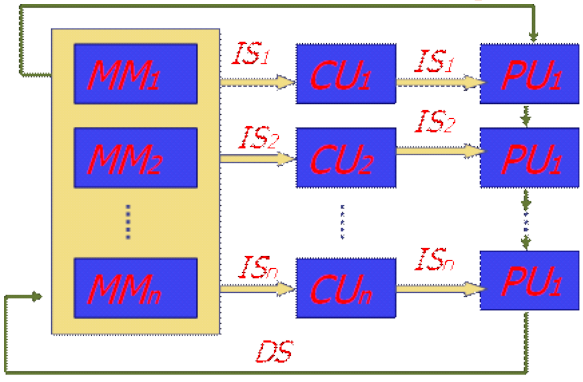
SISD计算机：传统的顺序处理机

2.单指令流多数据流SIMD(Single IS Multiple DS)

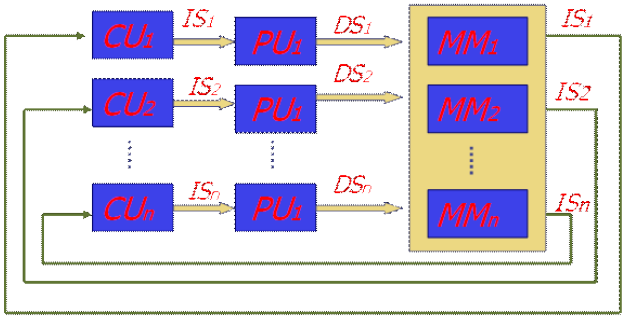
- 在SIMD计算机中，在同一部件（CU）的控制下，多个处理部件（PU）同时执行同一条指令所规定的操作，分别对各自的数据进行处理。



3.多指令流单数据流MISD(Multiple IS Single DS)



4.多指令流多数据流MIMD(Multiple IS Multiple DS)



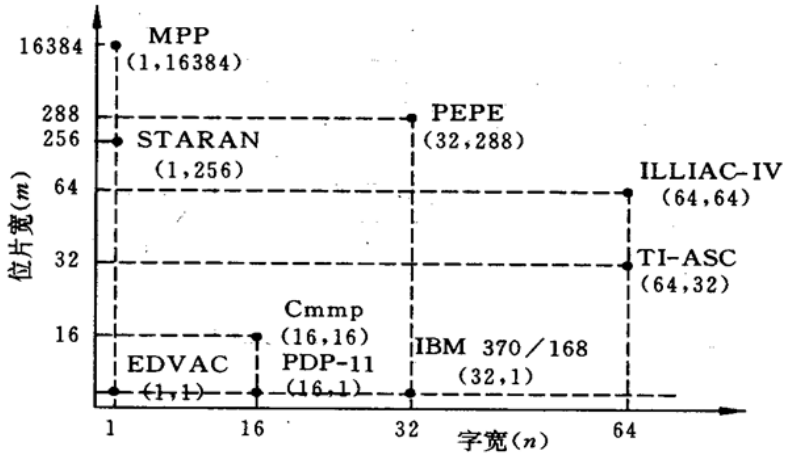
MIMD计算机：多处理机

- 1.1.4.2冯氏分类法
  1. 美籍华人冯泽云于1972年提出用最大并行度对计算机系统结构进行分类。
  2. 最大并行度**Pm**定义为：计算机系统在单位时间内能够处理的最大的二进制位数。假定每个时钟周期**Δti**内能同时处理的二进位数为**Pi**，则**T**个时钟周期内平均并行度为：

$$Pa = \sum_{i=1}^{T} Pi \ \Delta ti \ /T$$

  3. 平均利用率=**Pa/Pm**
  4. 设字宽**n**,位片宽**m**

面积代表一个系统的最大并行度



按最大并行度的冯氏分类法

7.冯氏分类法按最大并行度分类（最大并行度(Pm)指单位时间内能处理的最大二进制位数）。

1.字串位串WSBS(Word serial and bit serial)

2.字并位串WPBS(Word parallel and bit serial)

3.字串位并WSBP(Word serial and bit parallel)

4.字并位并WPBP(Word parallel and bit parallel)

- 1.1.4.3汉德勒(Händler)分类法
  1. 根据并行度和流水线提出的一种分类法。
  2. t(系统型号)=(k,d,w)
     - 程序控制部件(PCU)的个数k;
     - 算术逻辑部件(ALU)或处理部件(PE)的个数d;
     - 每个算术逻辑部件包含基本逻辑线路(ELC)的套数w，如加法器位数
  3. t(系统型号)＝(k×k’,d×d’,w×w’)：有流水线
     - k’表示流水线中程控部件的个数

- d’表示指令流水线中算术逻辑部件的个数
- W’表示操作流水线中基本逻辑线路的套数。

4. 例子：Cray-1有一个CPU，12个相当于ALU或PE的处理部件，可以最多实现8级流水线，字长为64位，可以实现1-14位流水线处理。所以它的系统结构可以表示为：
   t(Cray-1)＝(1, 12×8, 64×（1∽14）)

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

- 1.2.1.1 加快经常性事件的速度
- 加快经常性事件的速度是计算机设计中最重要也最广泛采用的设计准则。使经常性事件的处理速度加快能明显提高整个系统的性能
- 例如：在CPU中进行相加运算时，相加结果可能出现溢出现象，也可能无溢出发生，而经常出现的事件是不发生溢出的情况，而溢出是偶然发生的事件因此，在设计时应优化不发生溢出的情况，而对溢出处理则不必过多考虑优化。因为发生溢出的概率很小，不会对系统性能产生很大的影响。

- 1.2.1.2 阿姆达尔(Amdahl)定律

$$\text{加速比} = \frac{\text{采用改进措施后的性能}}{\text{没有采用改进措施前的性能}}$$

$$= \frac{\text{没有采用改进措施前执行某任务的时间}}{\text{采用改进措施后执行某任务的时间}}$$

$$\frac{T_0}{T_n}$$

- Amdahl定律认为：系统中改进某一部件对整个系统性能的提高与这种部件的使用频率或占总执行时间的比例有关。

$$Fe = \frac{\text{可改进部分占用的时间}}{\text{改进前整个任务的执行时间}}$$

$$Se = \frac{\text{改进前改进部分的执行时间}}{\text{改进后改进部分的执行时间}}$$

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

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

- 例1.1：假设将某系统的某一部件的处理速度加快到10倍,但该部件的原处理时间仅为整个运行时间的40%，则采用加快措施后能使整个系统的性能提高多少？
  解：由题意可知：Fe=0.4，Se=10，根据Amdahl定律

$$S_n = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

- 例1.2：假设FP指令集中浮点平方根操作FPSQR占整个测试程序执行时间的20%(fe0)，采用2种不同方法进行改进：
  一种方法是采用FPSQR硬件，使FPSQR操作的速度加快到10倍(Se0)。另一种实现方法是使所有浮点数据指令的速度加快，使FP指令的速度加快到1.6倍(Se1)，FP指令占整个执行时间的50%（Fe1)。请比较这两种设计方案。采用哪种实现技术来求浮点数平方根FPSQR的操作对系统的性能影响较大。
  解：分别计算出这两种设计方案所能得到的加速比：

$$S_{FPSQR} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$S_{FP} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

- 结论：
  Amdahl定律表示：
  Se→∞　Sn=1/(1-Fe)
  性能改善极限受Fe值的约束

- 1.2.1.3 CPU性能公式
  1.CPU的执行时间取决于三个要素
  （1）指令条数IC；**instruction count**
  （2）每条指令所花的时钟周期数CPI；　　**cycle per instruction**
  （3）时钟频率f(或时钟周期t)。

  2.公式推导
- 一个程序所花的CPU时间(T)可以用两种方式来表示：
- 每条指令的平均时钟周期数：
  CPI=执行程序所需的时钟周期数/IC
  所以：
  CPU执行时间=CPIxICx时钟周期时间
  其中：
  （1）时钟周期时间：器件工艺有关；
  （2）**CPI:**取决计算机组成和指令系统结构；
  （3）**IC:**取决于指令系统的结构和编译技术。
- 假定：　I_i是第i种指令的执行次数
  I_i/IC是第I种指令所占比例

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

$$CPU\text{时间(T)} = \text{时钟周期长度} * \sum_{i=1}^{n}(CPI_i * I_i)$$

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i * I_i)}{IC} = \sum_{i=1}^{n}(CPI_i * I_i / IC)$$

3.例1.3
- 如果FP操作的比例为25%，FP操作的平均CPI=4.0，其它指令的平均CPI为1.33，FPSQR操作的比例为2%，　FPSQR的CPI为20。
  假设有两种设计方案，分别把FPSQR操作的CPI和所有FP操作的CPI减至2。
  试利用CPU性能公式比较这两种设计方案哪一个更好(只改变CPI而时钟频率和指令条数保持不变)。
- 解：原系统的$CPI_{原}$= 25%×4 + 75%×1.33 = 2
  方案1：
  $CPI_{现} = CPI_{原} - 2\% \times (20 - 2) = 1.64$
  方案2：

$CPI_{现}=CPI_{原}-25\%\times(4-2)=1.5$

我们也可以根据以下公式计算出方案2系统的CPI

$CPI_{现}=75\%\times1.33+25\%\times2=1.5$

显然，提高所有FP指令处理速度的方案要比提高FPSQR处理速度的方案要好。

方案2的加速比 $=2\div1.5=1.33$

4.例1.4

- 假设某两台机器的指令系统中，执行条件转移指令需2个时钟周期，而其它指令只需1个时钟周期。

  CPU_1先用一条比较指令设置条件码，然后由转移指令测试条件码，决定是否转移，实现一次条件转移要执行比较和测试两条指令。如条件转移指令占总指令数的30%，每条转移指令都伴随一条比较指令，所以比较指令也将占30%。

  CPU_2采用比较功能和判别是否实现转移功能合在一条指令的方法，这样实现一条件转移就只需一条指令就可以完成。由于CPU_1在转移指令中不包含比较功能，因此它的时钟周期就可以比CPU_2要快1.35倍。

  现在要问，采用不同转移指令方案的CPU_1和CPU_2，那个工作速度会更快些？

$$CPI=\frac{\sum_{i=1}^{n}(CPI_i*I_i)}{IC}=\sum_{i=1}^{n}(CPI_i*I_i/IC)$$ 比例

- 解：$CPI_1=0.3\times2+0.7\times1=1.3$

  $T_{CPU1}=IC_1\times1.3\times t_1$

  根据假设，有 $t_2=1.35\times t_1$，CPU_2没有独立的比较指令，所以在CPU_2编的程序为CPU_1的70%，分支指令比例30%/70%=42.8%，它们2个时钟周期，其余的指令1个周期，占57.2%

  $CPI_2=0.428\times2+0.572\times1=1.428$

  CPU_2中没有比较指令，因此$IC_2=0.7\times IC_1$。

  $t_2=1.35\times t_1$

  $T_{CPU2}=IC_2\times CPI_2\times t_2=0.7\times IC_1\times1.428\times1.35\times t_1$

  $=1.349\times IC_1\times t_1$

  T_{CPU1}比T_{CPU2}小，CPU_1比CPU_2运行得更快些。

5.例1.5

- 在例1.4中，如果CPU_1的时钟周期只比CPU_2的快1.15倍，那么哪一个CPU会工作得更快些？
- 因$t_2=1.15t_1$，$T_{CPUA}=1.15\times IC_1\times t_1$，所以

  $T_{CPU2}=0.7\times IC_A\times1.428\times1.15\times t_1$

  $=1.15\times IC_1\times t_1$

  CPU_2所需时间较少，CPU_2比CPU_1运行得更快。

1.2.1.4 访问的局部性原理

- 局部性分时间上的局部性和空间上的局部性。

- 时间上的局部性是指最近访问过的代码也可能是不久将被访问的代码。
- 空间上的局部性是指那些地址上相邻近的代码可能会被一起访问。
- 存储系统的构成就是以访问的局部性原理为基础的。

1.2.1.5 采用并行性

- 采用并行性是改善计算机性能的重要方法。
- 系统级并行，例如多处理器和多磁盘技术，可以实现更大的吞吐量，而处理器和磁盘都可增加数目，对服务器来说具有重要价值。
- 单一处理器层面，一般采用指令级并行来提高性能，指令级并行的代表技术是流水线。
- 在数字电路设计层面可以发掘并行性，比如，组相联的Cache可以同时使用多个存储块，先行进位链能够加速求和的过程。

1.2.2 计算机系统设计者的主要任务

1.2.2.1 根据用户要求进行需求分析

- 体系结构的设计者要设计满足价格、供电、性能和可用性指标的计算机，首先要确定需求：

  1 应用领域 ： 是专用还是通用？是面向科学计算还是面向商用处理？是桌面计算机，还是服务器或嵌入式计算机？

  2 软件兼容层次 ： 如要求在程序设计语言层兼容，或者要求在目标代码层兼容等。

  3 操作系统需求： 如地址空间大小、存储管理、保护等。

  4 标准 ：如浮点数标准、I／O总线标准、网络标准、操作系统类型、程序设计语言标准等。

1.2.2.2 进行软硬件平衡

- 软硬件实现在功能上是等效的。但软件与硬件实现的特点不同：
- 硬件实现：速度快、成本高；灵活性差、占用内存少
- 软件实现：速度低、复制费用低；灵活性好、占用内存多
- 软硬件发展趋势：
- 硬件实现的比例越来越高
- 硬件所占的成本越来越低
- 软硬件的界面在上升

1.2.2.3 设计出符合今后发展方向的系统结构

系统结构的设计者还应关注实现技术和计算机应用方面的重要发展趋势。

这些发展趋势不仅影响机器未来的成本，也影响到所设计的系统结构的发展周期

这些发展趋势主要包括：

- 实现技术的发展趋势
- 集成电路功耗的发展趋势
- 成本的发展趋势

1.2.3 计算机系统设计的主要方法

1.2.3.1 由上往下设计

1. 设计过程如下：

- 面向应用的数学模型
- 面向应用的高级语言

- 面向这种应用的操作系统
- 面向操作系统和高级语言的机器语言
- 面向机器语言的微指令系统和硬件实现

2. 应用场合：专用计算机的设计(早期计算机的设计)
3. 特点：适用于所面向的应用领域，随着通用计算机价格降低，目前已经很少采用

1.2.3.2 由下往上设计

1. 设计过程如下：
   - 根据当时的器件水平，设计微程序机器级和传统机器级。
   - 根据不同的应用领域设计操作系统、汇编语言、高级语言编译器等。
   - 最后设计面向应用的虚拟机器级。
2. 应用场合：在计算机早期设计中(60～70年代)广为采用
3. 特点：容易使软件和硬件脱节，整个计算机系统的效率降低。

1.2.3.3 由中间开始设计

1. 设计过程如下：
   - 首先定义软硬件的分界面，包括指令系统、存储系统、输入输出系统、中断系统、硬件对操作系统和编译系统的支持等
   - 然后各个层次分别进行设计：软件设计人员设计操作系统、高级语言、汇编语言、应用程序等；硬件设计人员设计传统机器、微程序、硬联逻辑等
2. 应用场合：用于系列机的设计
3. 特点：软硬件的分界面在上升；硬件价格下降，软件价格上升；软硬件人员结合共同设计 优点
4. 由中间层开始设计其实就是由体系结构开始设计，是目前计算机的主要设计方法。

1.3 计算机系统结构的评价标准

评价一个计算机系统结构好坏的标准主要是性能和成本这两个指标。

1.3.1 性能

对用户来说，一个机器更好，通常是指程序运行的更快(桌面计算机)或者单位时间内完成的任务更多(服务器)。而对设计者来说，衡量机器性能的有效而可靠的标准就是实际程序的执行时间。

时间分为响应时间和CPU时间，要根据需要选择；

响应时间指完成任务的全部时间，包括磁盘、存储器、I/O设备访问和操作系统开销等；

CPU时间指任务在CPU上消耗的时间，不包括I/O等待和执行其它优先级更高任务的时间。

- 主要性能衡量标准
- 无论哪种标准，测试性能时比较的应该是相同负载的执行时间—程序和操作系统指令的总执行时间。

（1）MIPS

- 表示每秒百万指令条数。
- 对于一个给定的程序，MIPS 定义为：

$$MIPS = \frac{\text{指令条数}}{\text{执行时间} \times 10^6} = \frac{\text{时钟频率}}{CPI \times 10^6} = IPC \times Fz(M)$$

- 程序的执行时间为：

$$Te = \frac{\text{指令条数}}{MIPS \times 10^6}$$

- 越快的机器其MIPS越高
- 例1：已知 PentiumIV 2GHz 处理机的IPC＝4，计算它的指令执行速度。
  解：因为IPC＝4，Fz＝2000MHz
  因此，$MIPS_{PentiumIV2G}$＝Fz×IPC＝2000×4＝8000MIPS＝8GIPS
  即每秒钟80亿次(平均每秒钟执行80亿条指令)
- 主要优点：直观、方便。目前还经常使用
- 主要缺点：
  (1) 指令集不同时不准确，依赖于指令集合
  (2) 指令使用频度差别很大，依赖于程序
  (3) 可能和性能相反，如用软硬件分别实现浮点运算

（2）MFLOPS 每秒百万次浮点操作次数

$$MFLOPS = \frac{\text{程序中的浮点操作次数}}{\text{执行时间} \times 10^6}$$

- MFLOPS 仅仅只能用来衡量机器浮点操作的性能，而不能体现机器的整体性能。
- MFLOPS是基于操作而非指令的，所以它可以用来比较两种不同机器的浮点性能。
- MFLOPS依赖于操作类型。例如100%的浮点加要远快于100%的浮点除。
- MFLOPS只能作为一个参考标准。

（3）基准测试程序

- 测量性能的最好方法是通过实际的应用程序测试。这些方法包括：
- 从实际的程序中抽取少量关键循环程序段，并用它们来评价机器的性能。
- 小型基准测试程序。通常只有10-100行。
- 综合基准测试程序。模拟实际应用的特征和行为而编写的程序。
- 测试程序最终要给出性能评测报告。
- SPEC是最成功的基准测试程序集，有很多版本，各版本有不同的程序集，分为针对桌面计算机和服务器两种。
- 针对桌面计算机的测试程序又分为面向处理器和图形系统两种。
- 由于服务器功能的多样性，针对服务器的测试程序种类更多，但都以每秒钟处理的事务数作为标准。

- 性能评测结果的分析

1. 不同程序在不同机器上的执行时间
2. 平均执行时间

- 用平均执行时间代替总执行时间，平均执行时间是各执行时间的算术平均值。

$$Am = \frac{1}{n}\sum_{i=1}^{n} T_i \qquad \text{平均执行时间}$$

T_i: 第i个测试程序的执行时间，n是测试程序组中程序的个数

- 如果性能是用速度(例如MIPS和MFLOPS)表示，那么平均时间是调和平均。

$$Hm = \frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}} \qquad 其中Ri=1/Ti$$

- 这种方法无法表示程序集中单个程序的重要性。

3. 加权平均执行时间
   - 将权因子和执行时间的积相加，这叫做加权算术平均值。（所占比重）

$$Am = \sum_{i=1}^{n} W_i T_i \qquad 加权平均执行时间$$

   - 加权调和平均所体现的性能和加权算术平均相同。

$$Hm = \frac{1}{\sum_{i=1}^{n} \frac{W_i}{R_i}} \qquad 加权调和平均时间$$

   - 这种方法的难点在于权值的选择，因为这可能设计相关公司的利益。

4. 使用参考机器 将一个任务的执行时间标准化为一个参考机器的执行时间。
   - 参考机器的执行时间称为平均标准化时间。
   - 几何平均在反应性能上可以与机器无关，所以算数平均不可以用来求平均标准时间（与机器有关）。

$$几何平均数 G_m = \sqrt[n]{\prod_{i=1}^{n} R_i}$$

   - $R_i=1/T_i$ T_i: 第i个测试程序的执行时间， R_i: 是n个程序组成的工作负荷中第i个程序的速度， Π 表示连乘。

$$加权几何平均数 G_m = \prod_{i=1}^{n} (R_i)^{wi}$$

5. 标准偏差

1.3.2 成本

- 计算机系统的成本是指软件和硬件两方面的成本
- 当前软件成本由于其复杂性和长度的增加而不断增长，硬件成本则由于科技的发展尤其是VLSI的发展而快速下降。
- 计算机系统的成本主要包括以下两部分：
- 一次性开发成本；
- 每个部件的生产成本

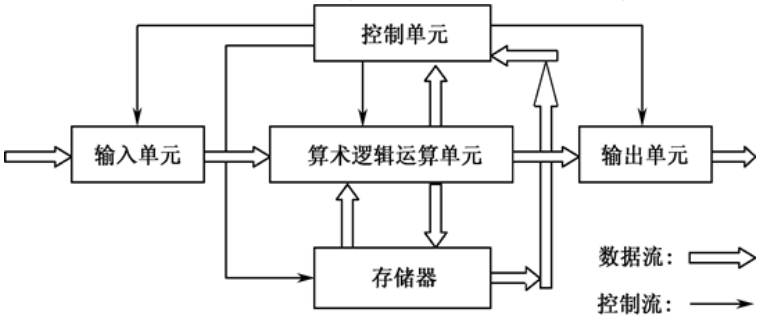
- 对于软件来说，其成本主要是指开发成本，软件的生产成本即复制费是很低的。
- 对于硬件，其成本既包括一次性开发成本，又包括生产成本，硬件的生产成本是远远高于软件的。
- 硬件成本的降低，让体系结构的设计者能够用复杂得多的硬件来构造系统，但并不意味着可以随意浪费硬件，要考虑技术含量和应用场合。

- 系统的高成本使得其商品竞争能力要降低。

1.4 计算机系统结构的发展

1.4.1 冯诺依曼体系结构及其改进

- 冯·诺依曼，美籍匈牙利科学家，参与了世界上第一台电子数字计算机“埃尼阿克”的设计工作
- 冯·诺依曼结构计算机，是最基本的结构，改进最多。



1.4.2 软件对系统结构的影响

- 软件兼容性(可移植性) 要求同一软件可不经修改或只须少量修改即可由一台机器移植到另一台机器上运行，即同一软件可应用于不同环境。
- 软件兼容性实现上主要有下面三种方法：
  - 采用系列机和兼容机方法
  - 系列机是同一个厂家生产的具有相同系统结构，不同的组成和实现技术的一系列不同型号的机器。
  - IBM系列机有370/115、125、135、145、158、168等一系列从低速到高速的各种型号。它们有相同的系统结构，但采用不同的组成和实现技术
    - 它们有相同的指令系统
    - 从程序设计者看，各档机器的字长都是32位
    - 各档机器都采用通道方式进行输入输出。
  - 兼容机是不同厂家生产的具有相同系统结构，不同的组成和实现技术的性价比不同的机器。
  - PC兼容机有8088、8086、80186、80286、80386、80486、Pentium、Pentium II、Pentium III等：
    - 不同工作主频；
    - 不同扩展功能：如浮点和多媒体功能等
    - 不同大小的Cache
    - 不同的字长：8位(8088)、16位(80286)、32位、64位。
  - 采用系列机和兼容机方法的主要优点：
    - 插件、接口等相互兼容；
    - 便于实现机间通信；
    - 便于维修、培训；
    - 有利于提高产量、降低成本。
  - 采用系列机和兼容机方法的主要缺点：限制了计算机系统结构的发展。
  - 几种兼容：

    √向上(下)兼容是指按某档次机器编制的程序，不加修改就能运行于比它高(低)档次的机器。

    √向前(后)兼容是指按某个时期投入市场的某种型号机器编制的程序，不加修改就

能运行于在它之前(后)投入市场的机器。

- 采用模拟与仿真方法
- 模拟是用软件方法在一台现有机器上实现另一台机器的指令系统。
- 仿真是用微程序直接解释另一台机器的指令系统的方法。
- 模拟方法速度低，仿真方法速度高，但仿真需要硬件支持，系统结构差别大的机器难于完全用仿真方法来实现。
- 通常把两种方法混合起来，尽可能用仿真的方法，无法仿真的指令再用模拟方法实现。

- 采用统一的高级语言方法
- 采用统一的高级语言是理想的实现软件兼容性的方法，可以无限制地在体系结构相同或完全不同的机器上进行软件移植。
- 但要统一高级语言，语言必须标准化，短期内难于实现。
- 20世纪70年代，有些试验性的高级语言计算机体系结构产生过，但复杂的实现反而影响了计算机性能的发展。

1.4.3 器件发展对系统结构的影响

- 系统结构的设计应考虑未来技术的发展，以延长计算机的生存周期，从长远发展考虑，必须了解计算机实现技术的快速变化。
- 四种实现技术
- 集成电路技术：晶体管密度每年增长约35%，芯片上的晶体管数目年增长40-55%。
- DRAM(动态随机存储器)：容量每年增长约40%
- 磁盘技术：90年以前，年增长30%，90后60%，96后100%，2004年降为30%，每比特成本比DRAM便宜50-100倍。
- 网络实现技术
- 由于计算机设计和生产需要一个周期，所以采用的实现技术必须有前瞻性，产品投产时应保证其成本或性能最优。
- 技术变化是连续的，但它的影响却是跳跃的。
- 当MOS技术发展到可以集成25000-50000个晶体管时，制造32位机才成为可能。
- 80年代末，技术进步使Cache在芯片内部出现，消除了Cache与处理器的芯片交叉，提高了性价比和性能功耗比。

- 性能发展趋势：带宽优于时延
- 带宽或吞吐量：在给定时间内所完成的工作总量
- 时延或响应时间：从事件开始到完成所需要的时间。
- 在近20到25年间：
  - 对微处理器：带宽提高了1000到2000倍，时延改进了20到40倍
  - 存储器和磁盘，带宽提高了120到140倍，时延改进了4到8倍
- 一种简单的法则是带宽的提高速度至少相当于时延改善速度的平方，计算机设计者应据此作出相应规划。

- 晶体管性能与连线的规模
- 集成电路的加工工艺是用特征尺寸来表示的，特征尺寸是晶体管或者连线在x或y方向的最小尺寸，。晶体管密度和特征尺寸大小的平方成反比。

- 一般而言，晶体管电路的性能随着特征尺寸的减少而线性增加。除了开关时延外，晶体管之间的连线也会产生时延，而且相对而言，连线时延是有更大的改进空间，也更为关键。
- 从1971年到2006年，特征尺寸从10微米降到了0.09微米。

1.4.4 应用对系统结构的影响

- 高结构化的数值计算
- 非结构化的数值计算
- 实时多因素问题
- 大存储容量和输入输出密集问题
- 图形学和设计问题
- 人工智能

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

- 所谓的并行性是指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。只要在时间上重叠，就存在并行性。它包括同时性与并发性两种含义。
  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
- 计算机系统中的并行性有不同的等级。从处理数据的角度看，并行性等级从低到高可分为：
  （1）字串位串：同时只对一个字的一位进行处理。这是最基本的串行处理方式，不存在并行性。
  （2）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。这里已开始出现并行性。
  （3）字并位串：同时对许多字的同一位进行处理。这种方式有较高的并行性。
  （4）全并行：同时对许多字的全部位进行处理。这是最高一级的并行。
- 从执行程序的角度看，并行性等级从低到高可分为：
  （1）指令内部并行：一条指令执行时各微操作之间的并行。
  （2）指令级并行：并行执行两条或多条指令。
  （3）任务级或过程级并行：并行执行两个以上过程或任务
  （程序段）
  （4）作业或程序级并行：并行执行两个以上作业或程序。

1.5.2 提高并行性的技术途径

计算机中提高并行性的措施多种多样，就是基本思想而言，可以归纳为如下3条途径：
（1）时间重叠。即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，以加快硬件周转时间而赢得速度。因此时间重叠可称为时间并行技术。
（2）资源重复。在并行性概念中引入空间因素，以数量取胜的原则，通过重复设置硬件资源，大幅度提高计算机系统的性能。随着硬件价格的降低，这种方式在单处理机中广泛使用，而多处理机本身就是实施“资源重复”原理的结果。
（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。例如多道程序、分时系统就是遵循“资源共享”原理而产生的。资源共享既降低了成本，又提高了计算机设备的利用率。

1.5.3 单机系统中并行性的发展

- 在发展高性能单处理机过程中，起着主导作用的是时间重叠原理。例如：解释指令的5个子过程分别需要5个专用部件，
- 即取指令部件(IF)、指令译码部件(ID)、指令执行部件(EX)、访问存储器部件(M)、写回结果部件(WB)。将它们按流水方式连接起来，就满足时间重叠原理，从而使得处理机内部同时处理多条指令，提高了处理机的速度。显然，时间重叠技术开发了计算机系统中的指令级并行。
- 在单处理机中，资源重复原理的运用也已经十分普遍。例如不论是非流水线处理机，还是流水线处理机，多体存储器和多操作部件都是成功应用的结构形式。在多操作部件处理机中，通用部件被分解成若干个专用操作部件，如加法部件、乘法部件、除法部件、逻辑运算部件等。一条指令所需的操作部件只要空闲，就可以开始执行这条指令，这就是指令级并行。
- 资源共享 虚拟打印机

1.5.4 多机系统中并行性的发展

- 多机系统也遵循时间重叠、资源重复、资源共享原理，
- 向着三种不同的多处理机方向发展。但在采取的技术措施上与单处理机系统有些差别。
- 紧耦合系统又称直接耦合系统，指计算机间物理连接的频带较高，一般是通过总线或高速开关实现计算机间的互连，可以共享主存。由于具有较高的信息传输率，因而可以快速并行处理作业或任务。
- 松耦合系统又称间接耦合系统，一般是通过通道或通信线路实现计算机间的互连，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。松耦合系统表现为两种形式：一种是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，供其他机器继续处理。另一种是计算机网，通过通信线路连接，以求得更大范围的资源共享
- 多处理机中为了实现时间重叠，将处理功能分散给各专用处理机去完成，即功能专用化，各处理机之间则按时间重叠原理工作。许多主要功能，如数组运算、高级语言编译、数据库管理等，也逐渐分离出来，交由专用处理机完成，机间的耦合程度逐渐加强，从而发展成为异构型多处理机系统。