

计算机系统结构

Computer Architecture

主 讲：刘超

中国地质大学（武汉）计算机学院

教材及主要参考书目

- 《计算机系统结构教程》（第二版）
 - 张晨曦等 编著
 - 清华大学出版社

其他参考书目

- 《计算机系统结构——量化研究方法》第5版
 - Hennessy, Patterson
- 《云计算与分布式系统（从并行处理到物联网）》
 - 黄凯等 机械工业出版社

主要内容、课时安排

- 第一章：计算机系统结构的基础知识
- 第二章：指令系统的设计
- 第三章：流水线技术
- 第四章：向量处理机
- 第五章：指令级并行及其开发——硬件方法
- 第六章：指令级并行及其开发——软件方法
- 第七章：存储系统 &&第八章：输入输出系统
- 第九章：互连网络
- 第十章：并行处理机与多处理机

考试、成绩

○ 考试

- 笔试（闭卷）

○ 成绩

- 笔试成绩占60%
- 平时成绩占40%（到课情况、作业、测试等）

课程特点、推荐的学习方法

○ 课程特点

- 概念多
- 比较抽象

○ 推荐的学习方法

- 把握脉络（根据教材大纲来学习）
- 掌握知识点（基本概念、基本公式）
- 独立完成习题（加深理解）

第一章 计算机系统结构基本概念

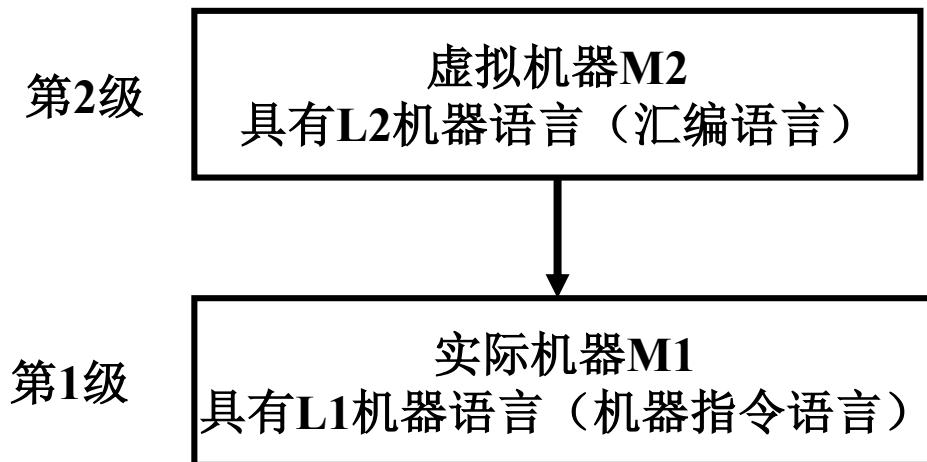
- 1.1 计算机系统结构的层次性与透明性
- 1.2 计算机系统结构、组成与实现
- 1.3 计算机系统结构的分类
- 1.4 计算机系统设计技术
- 1.5 计算机系统的性能评价
- 1.6 计算机系统结构的发展
- 1.7 并行系统结构概念
- 1.8 小结
- 1.9 习题

1.1 计算机系统结构的层次性与透明性

- 1) 层次结构的划分
- 2) 各机器级的实现技术
- 3) 透明性

1) 层次结构的划分

- **机器**：指能够存储和执行程序的算法和数据结构的集合体
- **实际机器**：由硬件和固件实现的机器。
 - 固件（Fireware）是一种具有软件功能的硬件
- **虚拟机器（Virtual Machine）**：以软件为主实现的、构筑在实际机器之上的机器。



语言和虚拟机之间存在对应关系

- 计算机语言可以分很多级

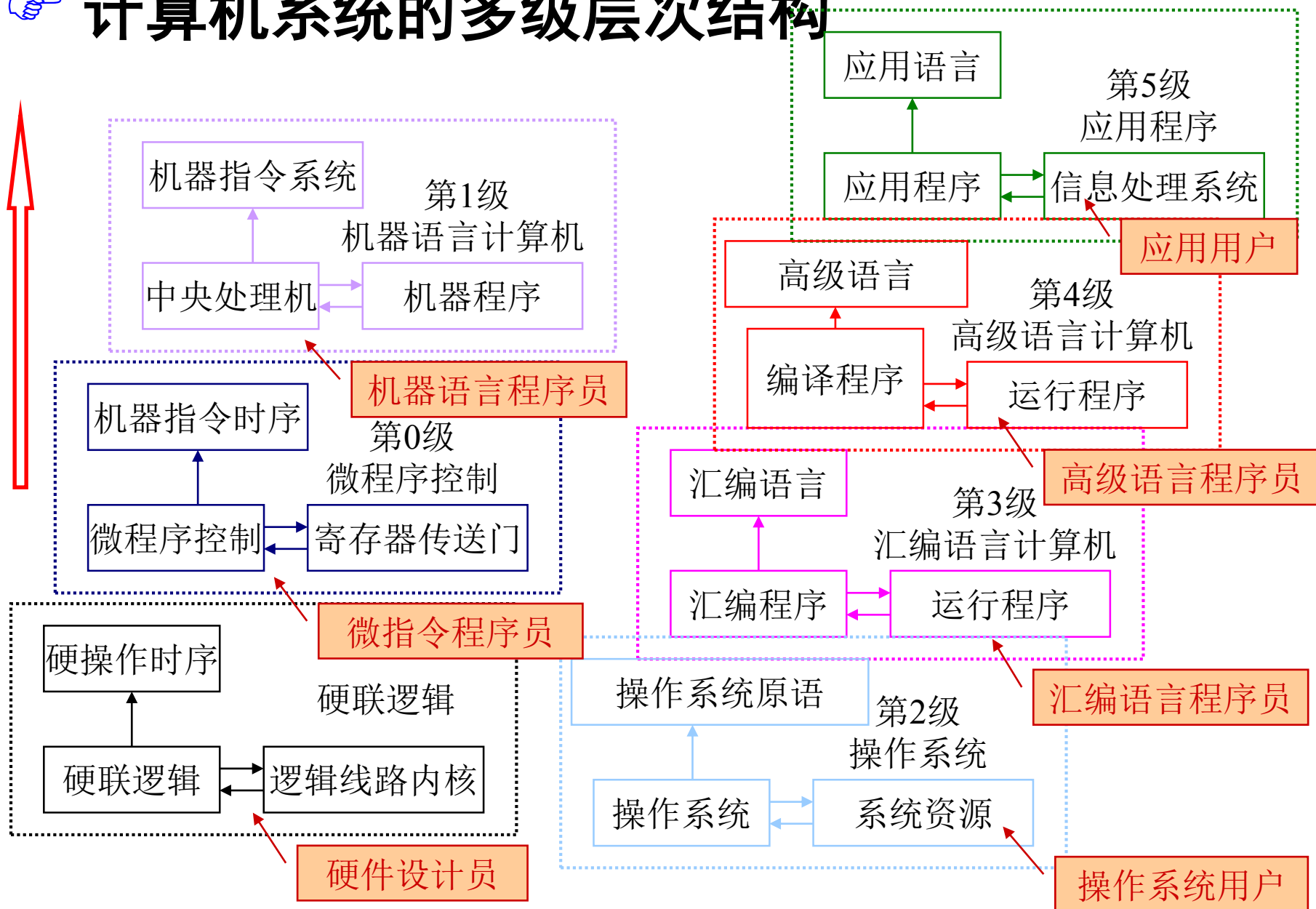
- 低级语言简单
- 高级语言复杂，功能强大

- 机器也可按照不同层次的语言划分层次

- 由低到高分别为微程序机器、传统机器、操作系统机器、汇编语言机器、高级语言机器、应用语言机器
- 对某一级的程序员来讲，都可以将此机器级看成一台独立的机器，可以使用相应机器级的语言
- 有 n 层不同的语言，就对应应有 n 层不同的机器。



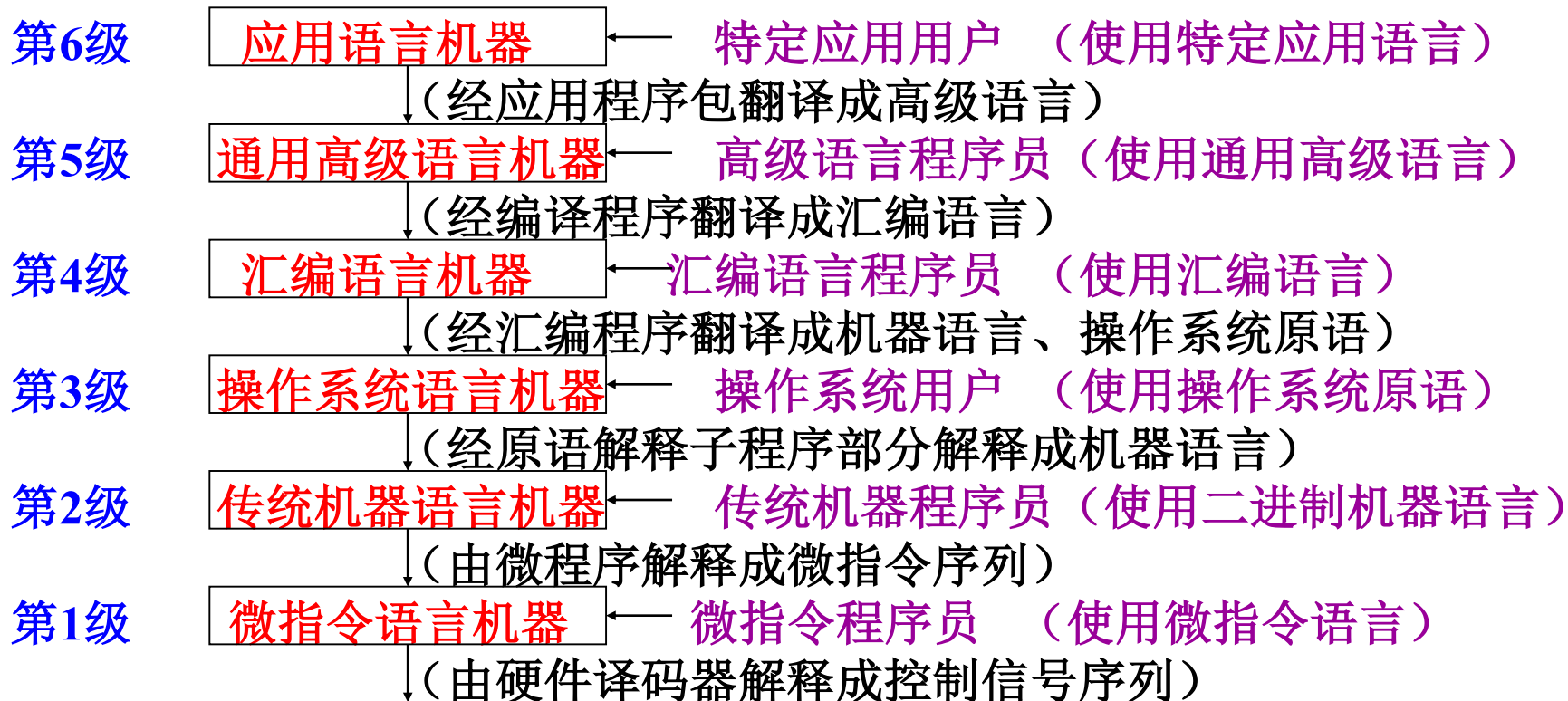
计算机系统的多级层次结构



2) 各机器级的实现技术

- 低层机器级为高层机器级的实现提供支持。实现的技术就是**翻译**(Translation)和**解释**(Intepretation)。
 - **翻译**：先用转换程序将高一级机器上的程序**整个地变换**成为低一级机器上可运行的等效程序，然后再在低一级机器级上去实现的技术。
 - **解释**：在低一级机器上用它的一串语句或指令来仿真高一级机器级上的一条语句或指令的功能，通过对高一级机器语言程序中的每条语句或指令**逐条解释**来实现的技术。
 - **解释执行比翻译执行所花的时间多，但占用的存储空间较少。**

多级层次模型的实现技术



○ 多级层次结构的作用

- 有利于正确理解计算机系统的工作，明确软件、硬件和固件在计算机系统中地位和作用
- 有利于正确理解各种语言的实质及其实现
- 有利于探索虚拟机器新的实现方法，设计新的计算机系统

3) 透明性

☞ 透明性现象

☞ 一种本来是存在的事物或属性，但从某种角度

看似不存在，称为透明性（Transparency）现象。

- 在计算机系统中，低层机器级的概念性结构和功能特性，对高级机器级的用户（高级语言程序员）来说是透明的。
- 从某一层的使用者角度来看，只需通过该层的语言就可以使用机器，而不需要关心其下层的机器级是如何工作和实现其功能的。

1.2 计算机系统结构、组成与实现

- 1) 计算机系统结构
- 2) 计算机组成与实现
- 3) 结构、组成与实现之间的相互影响
- 4) 计算机系统结构在计算机学科中的作用

1) 计算机系统结构

- **计算机系统结构**（Computer Architecture）是指计算机多级层次结构中**机器语言机器级的结构**，它是**软件和硬件/固件的主要界面**。是由机器语言程序、汇编语言源程序、高级语言源程序翻译生成的**机器语言目标代码能在机器上正确运行所应具有的界面结构和功能**。
- 计算机系统结构的**研究目标**：主要研究软件、硬件功能分配和确定软件与硬件之间的界面，即哪些由硬件实现，哪些由软件实现。

Computer Architecture - Definition

- **Computer Architecture = ISA + MO**

- **Instruction Set Architecture**

- What the executable can “see” as underlying hardware
- Logical View

- **Machine Organization**

- How the hardware implements ISA ?
- Physical View

计算机系统结构的含义

○ 计算机系统结构的定义

- 1964年，IBM360系列机的主设计师Amdahl提出了系统结构的定义：

计算机系统结构就是程序员所看到的计算机的基本属性，即硬件子系统的概念结构及其功能特性。

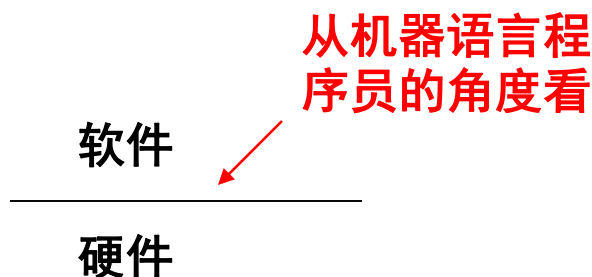
○ 争议焦点？

不同的层次观察者看到的特性不同，这里的程序员到底是指哪一级的程序员呢？

👉 Amdahl定义的系统结构特性

在Amdahl定义中的程序员主要指**机器语言**和**编译程序**设计者，一般认为系统结构特征包括以下几个方面：

- Ψ 指令系统
- Ψ 数据表示
- Ψ 操作数的寻址方式
- Ψ 寄存器的构成定义
- Ψ 中断机构和例外条件
- Ψ 存储体系和管理
- Ψ I/O结构
- Ψ 机器工作状态的定义和切换
- Ψ 信息保护



系统结构特性

- 1. **数据表示**（硬件能够直接识别和处理的数据类型和格式等）；
- 2. **寻址方式**（包括最小寻址单位、寻址方式的种类、表示和地址计算等）；
- 3. **寄存器组织**（包括各种寄存器的配置数目和功能定义）；
- 4. **指令系统**（包括机器指令的操作类型和格式、指令间的排序方式和控制机构等）；
- 5. **存储系统**（包括编址方式、存储容量、最大编址空间等）；
- 6. **中断机构**（中断源的分类管理和中断服务功能设计）；
- 7. **机器工作状态**（如管态、目态等）的定义和切换；
- 8. **输入/输出子系统**结构与管理；
- 9. **信息保护手段**及其实现。

系统结构的研究范围

- **特性**——指令系统、数据表示、寻址方式、寄存器集等
- **界面设计**——确定硬件功能。
- **新型系统结构设计**——并行性、数据流、推理机、神经网络
- **性能成本评价**——运算速度、存储容量、I/O带宽

2)计算机组成与实现

- **计算机组成**（Computer Organization）：
是计算机系统结构的**逻辑实现**，主要研究硬件系统在逻辑上是如何组织的。包括机器内部的数据流和控制流的组成以及**逻辑设计**等。
- 着眼于机器内各事件的排序方式与控制机构、各部件的功能以及各部件之间的联系。



○ 计算机组成设计要确定：

- 数据通路宽度
- 专用部件的设置
- 各种操作对部件的共享程度
- 功能部件的并行度
- 控制机构的组织方式
- 缓冲、排序技术
- 预测、预判技术
- 可靠性技术

- **计算机实现**（Computer Implementation）：是指计算机组成的**物理实现**，主要**着眼于器件技术和微组装技术**。包括处理机、主存等部件的物理结构，器件的集成度和速度，线路划分与连接，制造技术与工艺等。
- **器件技术**的发展在计算机实现技术中起着主导作用。

三个层次

- 计算机系统结构——特性设计
- 计算机组成——逻辑设计
- 计算机实现——物理设计，如器件选择，机械、封装、印板、机箱、电源、冷却设计

3)结构、组成与实现之间的相互影响

- 具有相同系统结构的计算机可以采用不同的组成。
 - 例如具有相同指令系统的计算机，指令的取出、译码、取操作数、运算、存结果可采用顺序方式进行解释，也可采用流水线方式在时间上重叠来提高速度。
 - 再如：乘法指令可以利用专用乘法器来实现，也可以通过加法器重复相加、移位来实现。



○ 一种计算机组成可以有多种不同的实现方法。

- 取决于计算机系统性能和价格的要求以及器件技术的发展情况。
- 例如：主存器件的选择上，可选择TTL型器件，也可以采用MOS器件；既可以采用单片VLSI集成电路，也可采用多片LSI或MSI集成电路组成；可选用高速芯片，也可选择低速芯片。

○ 反过来，计算机实现和计算机组成也会影响计算机系统结构。

- 例如：由于器件技术的发展，系统结构由大型机向中、小型甚至微型机上迁移的速度加快。
- 系统结构的设计应尽量减少对各种组成技术和实现技术的限制，同一种系统结构，应该既能够在高档机上用复杂的、较贵的组成技术实现，也可在低档机上采用简单的、便宜的组成实现。例如IBM370系列机具有相同的系统结构，不同档次的机器，其组成和实现都不同。

○ 三者不同，但又有密切联系。随着时间推移，三者之间界限越来越模糊。例如，现在已经将功能模块设计移入到系统结构的考察范畴。

○ 练习：对于计算机系统结构，下列哪些是不透明的？

- 存储器的模 m 交叉存取；浮点数据表示；I/O系统是采用通道还是外围处理机方式；数据总线宽度；字符行运算指令；阵列运算部件；通道是采用结合型还是独立型；单总线结构；访问方式保护；程序性中断；串行、流水控制方式；堆栈指令；存储器最小编址单位；Cache存储器。
- 解答：对于计算机系统结构不透明的包括：浮点数据表示；I/O系统是采用通道还是外围处理机方式；字符行运算指令；访问方式保护；程序性中断；堆栈指令；存储器最小编址单位。

系统结构的例子

系统结构	产品
Digital Alpha (V1, V3) 1992-97	DEC21064, 21164, 21264
HP PA-RISC (V1.1, V2.0) 1986-96	HP3000(930,950), HP9000(800,850) PA7100, PA8000
Sun Sparc (V8, V9) 1987-95	TI SuperSPARC TMS390Z50 (in Sun SPARCstation 20)
MIPS	MIPS 2000, 3000, 4000, 8000, 10000(in SGI workstation)
IBM PowerPC	PPC750, 740, 604, 603, 601, RS/6000
Intel IA-32, IA-64 1978-96	I386, I486, P, PII, PIII, P4, Itanium
AMD x86-64	SledgeHammer

1.3 计算机系统的分类

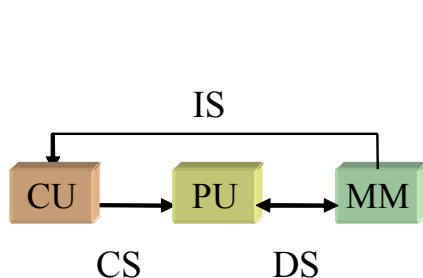
○ 1、Flynn（弗林）分类法：

- 按照指令流和数据流的多倍性状况对计算机系统进行分类。
- 指令流(Instruction Stream):机器执行的指令系列。
- 数据流(Data Stream):由指令流调用的数据序列,包括输入数据和中间结果。
- 多倍性(Multiplicity):在系统最受限制的部件(瓶颈)上同时处于同一执行阶段的指令或数据的最大可能个数。

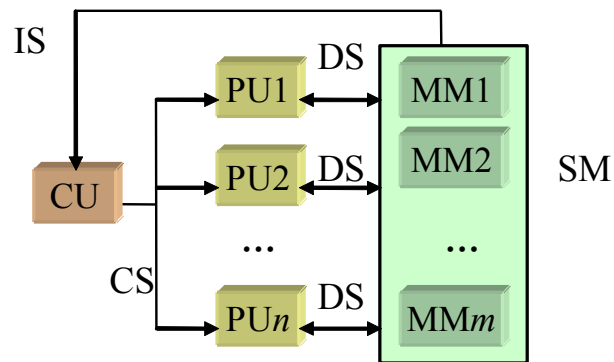
- **SISD 单指令流单数据流**，代表是传统的数字处理机。如:IBM 370,VAX 11/780,Intel 80X86;
- **SIMD 单指令流多数据流**，代表是阵列处理机或并行处理机。如: ILLIAC-IV,CRAY-1,YH-1,CYBER-205;
- **MISD 多指令流单数据流**，代表:退耦计算机及脉动机，对此类型有争议，有人将VLIW归入此类;
- **MIMD 多指令流多数据流**，代表是多处理机。如: IBM 370/168, CRAY, YH-2;

Flynn分类法各类机器结构示意图

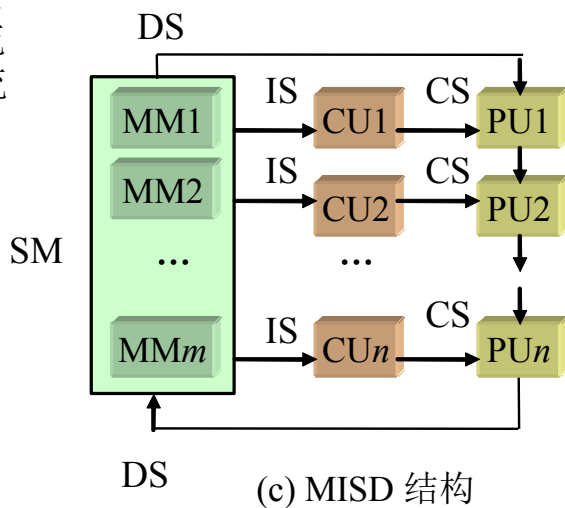
CU : 控制部件
PU : 处理机
MM : 主存模块
SM : 共享主存
IS : 指令流
CS : 控制流
DS : 数据流



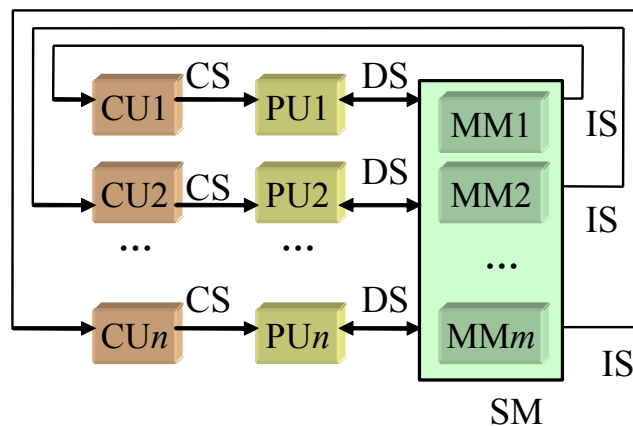
(a) SISD 结构



(b) SIMD 结构



(c) MISD 结构



(d) MIMD 结构

图 Flynn 分类法中四种系统的基本结构

Flynn计算机分类举例

类型	计算机的型号
SISD	IBM 370, VAX 11/780, MC 680X0, INTEL 80X86
SIMD	ILLIAC-IV, ICL-DAP, CRAY-1, YH-1 (银河1), CYBER-205, CM-2
MIMD	IBM 370/168, C_{mmp} , CRAY X-MP, YH-2(银河2)

2、冯氏分类法

- 1972年，美籍华人冯泽云教授提出，用数据处理的最大并行度(P_m)对计算机系统结构进行分类。
- **最大并行度**：计算机系统在单位时间内能够处理的最大的二进制位数。
 - n ：一个字中同时处理二进制的位数；
 - m ：一个位片或功能部件中能同时处理的字数。

○ 字串位串 (WSBS)

- $n=1$, $m=1$, 位串处理方式, 每次只处理一个字中的一位, 如第一代纯串行计算机。

○ 字串位并 (WSBP)

- $n > 1$, $m=1$, 字 (字片) 处理方式, 每次处理一个字中的 n 位, 如传统位并行单处理机。

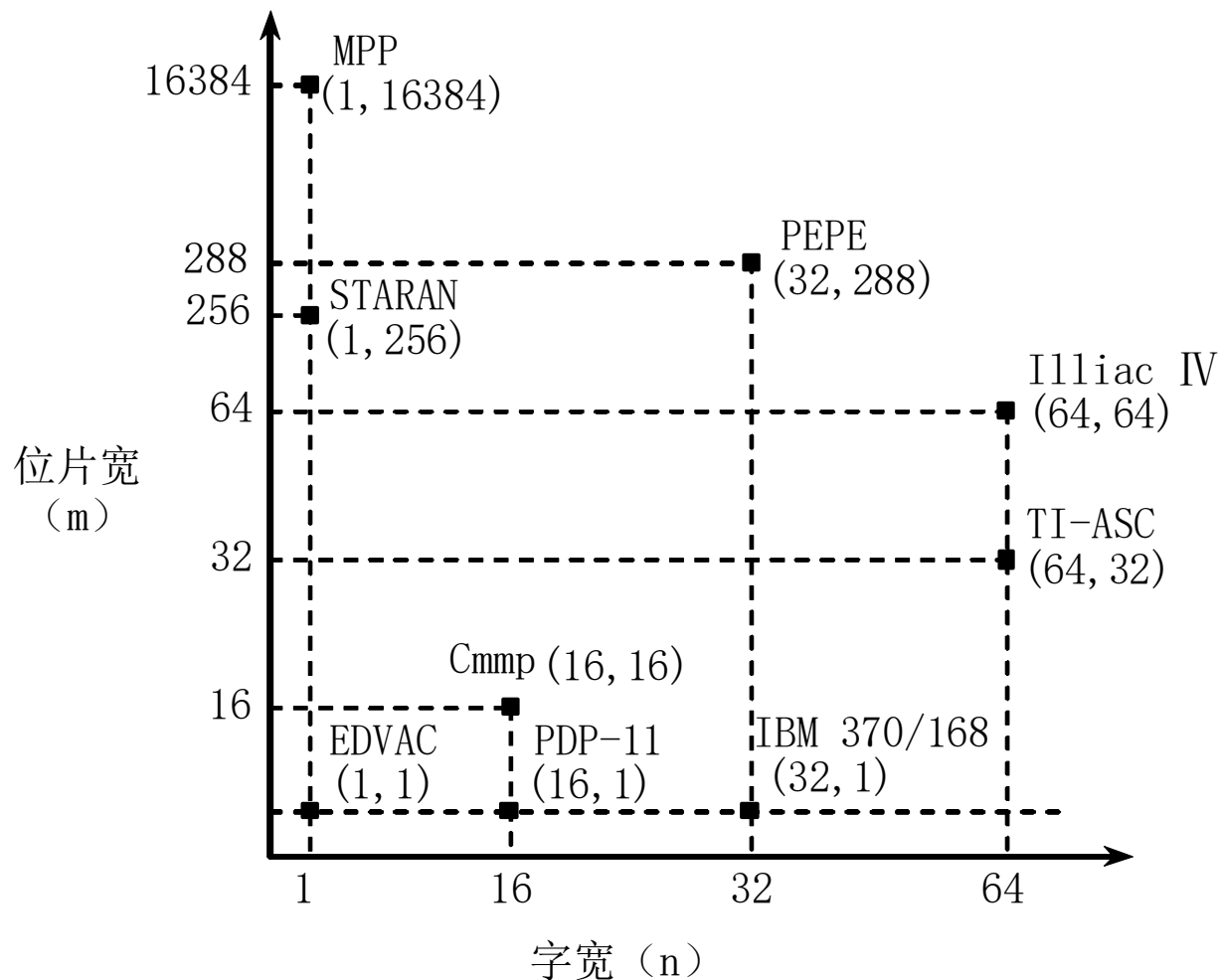
○ 字并位串 (WPBS)

- $n=1$, $m > 1$, 位 (位片) 处理方式, 每次处理 m 个字中的一位, 如某些相联处理机及阵列处理机。

○ 字并位并 (WPBP)

- $n > 1$, $m > 1$, 全并行处理方式, 一次处理 m 个字, 且每个字为 n 位, 如某些相联处理机, 大多数阵列处理机及多处理机。

- 用平面直角坐标系中的一个点代表一个计算系统，其横坐标表示字宽（ n 位），纵坐标表示一次能同时处理的字数（ m 字）。 $m \times n$ 就表示了其最大并行度。



○ 平均并行度

- 与最大并行度 P_m 密切相关的一个指标。
- 取决于系统的运用程度，与应用程序有关。

- 假设每个时钟周期内能同时处理的二进制位数为 P_i ，则 T 个时钟周期内的平均并行度为：

$$P_a = \frac{\sum_{i=1}^T P_i}{T}$$

- 系统在 T 个时钟周期内的平均利用率定义为：

$$\mu = \frac{P_a}{P_m} = \frac{\sum_{i=1}^T P_i}{TP_m}$$

3、Händler分类法

- 1977, 德国Händler (汉德勒) 提出, 按照计算机系统的并行程度和流水处理程度提出的分类法。
- 将计算机的硬件结构分成三个层次: 程序控制器PCU、算逻部件ALU或处理部件PE、基本逻辑电路ELC。
- 一个计算机系统可用三对整数加以描述:

$$T(C)=<K \times K', D \times D', W \times W'>$$

- 其中:

K=计算机中的程序控制器 (PCU)的个数

K'=宏流水线中的程序控制器(PCU)数

D=每个PCU所控制的ALU(或PE)数

D'=指令流水线中的ALU(或PE)数

W= 每个算逻部件所包含的基本逻辑线路(ELC)的套数

W'=操作流水线中基本逻辑线路(ELC)的套数

例1：CDC6600计算机系统有一个CPU, ALU有10个功能部件可连成一条10级流水线，字长为60位。此外，有10个可并行工作的外围I/O处理器，每个I/O处理器，有一个ALU，字长12位。

$$\begin{aligned} \text{则： } T(\text{CDC6600}) &= T\langle \text{中央处理器} \rangle \times T\langle \text{I/O处理器} \rangle \\ &= \langle 1, 10 \times 10, 60 \rangle \times \langle 10, 1, 12 \rangle \end{aligned}$$

例2：CRAY-1是一个单CPU计算机系统，有12个相当于ALU或PE的处理部件，最多可实现8个处理部件的流水，字长为64位，可以实现1~14位流水处理。

$$\text{则： } T\langle \text{CRAY-1} \rangle = \langle 1, 12 \times 8, 64 \times (1 \sim 14) \rangle$$

● 几个例子:

$$t(\text{PDP-11}) = (1, 1, 16)$$

$$t(\text{Illiac IV}) = (1, 64, 64)$$

$$t(\text{STARAN}) = (1, 8192, 1)$$

$$t(\text{Cmmp}) = (16, 1, 16)$$

$$t(\text{PEPE}) = (1 \times 3, 288, 32)$$

$$t(\text{TI-ASC}) = (1, 4, 64 \times 8)$$

1.4 计算机系统设计技术

- 1) 计算机系统设计的定量原理
- 2) 计算机系统设计的基本方法

1) 计算机系统设计定量原理

- **加速比(Speedup Ratio)**: 衡量系统采用的改进措施对系统性能提高的程度。
- 加速比 : $S_p = T_0 / T_e$
 - T_0 : 改进前执行某一任务所需时间
 - T_e : 改进后执行某一任务所需时间
 - 一般 $T_0 > T_e$, 所以 $S_p > 1$

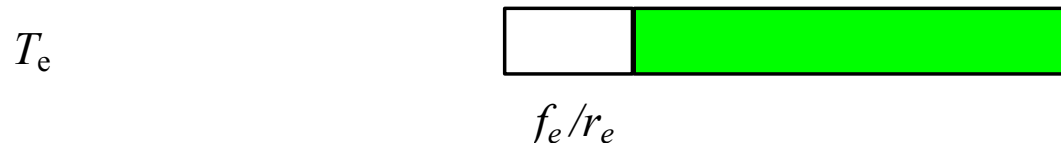
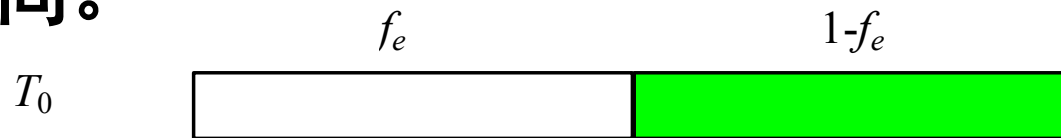
Amdahl定律

- **Amdahl定律**：系统中某一部件采用某种更快改进方式后所能获得的系统性能改进程度，取决于该部件**被使用的频率**，以及该部件在**改进前与改进后执行时间的加速比**。
- f_e 表示被改进部分改进前的执行时间在总执行时间 T_0 中所占的百分比($0 \leq f_e < 1$)，也就是被改进部件被使用的频率。
- r_e 表示被改进部分改进前的执行时间与改进后的执行时间的比值，即被改进部分的单独加速比。 ($r_e > 1$)
- 那么，改进后整个系统的加速比为：

$$S_p = \frac{T_0}{T_e} = \frac{1}{(1 - f_e) + f_e / r_e}$$

Amdahl定律的推导过程

- 部件改进后，系统的总执行时间等于不可改进部分(占 $1-f_e$)的执行时间加上可改进部分(占 f_e)改进后的执行时间。



$$S_p = \frac{T_0}{T_e} = \frac{f_e T_0 + (1 - f_e) T_0}{\frac{f_e}{r_e} T_0 + (1 - f_e) T_0} = \frac{1}{(1 - f_e) + f_e / r_e}$$

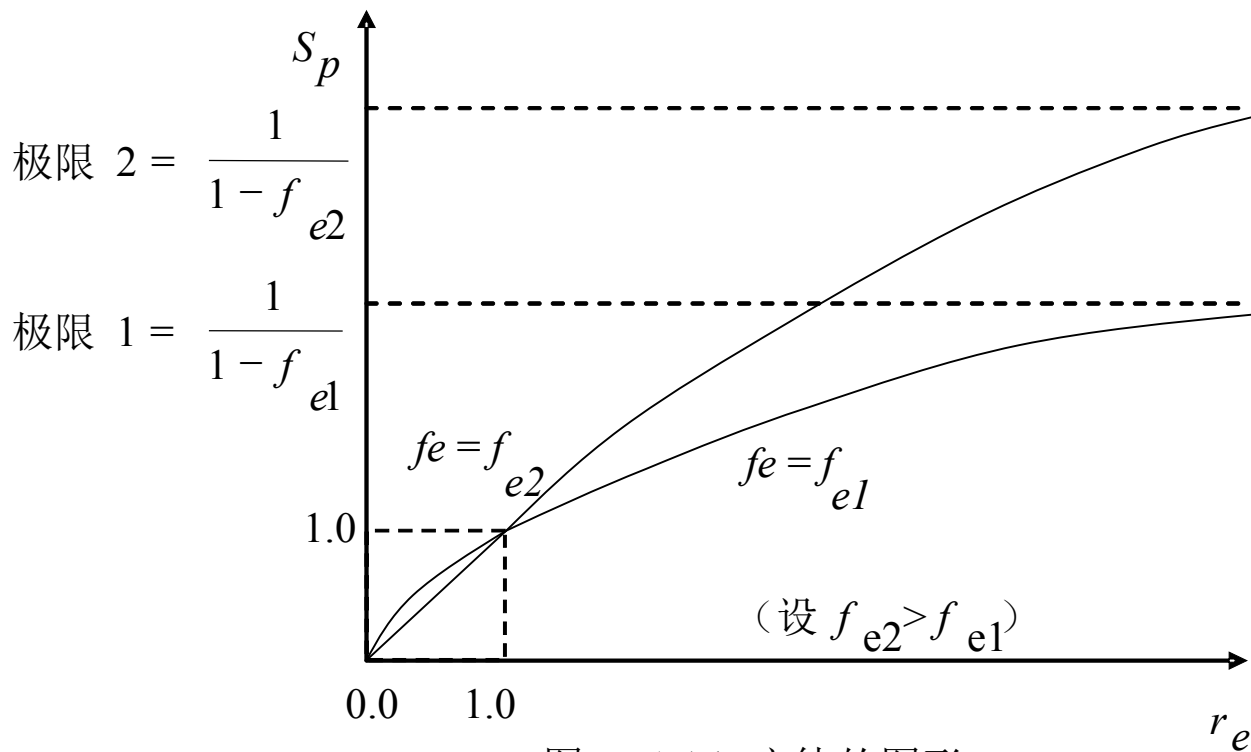
Amdahl定律推论

- 推论：系统性能提高程度 S_p 受到被改进部分执行时间所占比例 f_e 的限制，即使被改进部分的改进效果非常好，使 $r_e \rightarrow \infty$ ，获得整个系统性能改进的极限值 $S_p = 1/(1-f_e)$ ，仍受到 f_e 的限制。可见，具有高性能价格比的计算机系统应是一个整体性能优化平衡的系统，而不应该追求个别部件的高性能。

$$S_p = \frac{T_0}{T_e} = \frac{1}{(1-f_e) + f_e/r_e} \quad \text{若 } r_e \rightarrow \infty \quad S_p = \frac{1}{(1-f_e)}$$

Amdahl定律的图形

从图上可以看出，增大 r_e 和 f_e 对 S_p 都有提升作用；但当 f_e 固定时，一味增大 r_e 对 S_p 的作用会越来越不显著。



由图中曲线可知，为使整个系统能获得较高性能加速比，则可改进部分必须占有较大的比例；否则，改进该部分的功能就没有多大意义。

图 Amdahl 定律的图形

计算机系统设计的定量原则

- 由Amdahl定律得出计算机系统设计的重要定量原则：
 - 加速频繁出现的事件(Make the common case fast)，是最重要和最广泛采用的计算机设计准则。
 - 程序访问的局部性原理是指程序执行过程中，呈现出频繁重新使用那些最近已被使用过的数据和指令的规律。统计表明，一个程序用90%的执行时间去执行仅占10%的程序代码。程序访问的局部性包括时间局部性和空间局部性。
 - 程序访问的局部性原理为通过改进频繁出现的事件来提高整个系统的性能提供了可能。如在层次存储体系中，大部分程序存储在慢速部件中，而少数程序存放在虚存、Cache高速缓存中，而整个层次存储体系性能接近Cache。

程序访问局部性原理

- **程序访问局部性原理**：程序往往频繁重复使用它刚刚访问过的数据和指令。
 - 原因：程序的顺序指令和程序的循环等。
 - 分类：
 - **时间局部性**：近期被访问的代码，很可能不久又将再次被访问。
 - **空间局部性**：是指地址上相邻近的代码可能会被连续的访问。
 - 为解决计算机系统设计过程中，高性能和高成本之间的矛盾提供了解决方法。目前，计算机指令系统设计和存储器结构设计都是建立在程序访问局部性这一规律基础上的。

例：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占整个处理时间的40%，则采用加快措施后能使整个系统的性能提高多少？

解： $f_e=0.4$, $r_e=10$, 据Amdahl定律,

$$S_p=1/(0.6+0.4/10)=1/0.64\approx1.56$$

例：假设FPSQR操作占整个测试程序执行时间的20%。一种实现方法是采用FPSQR硬件，使FPSQR操作的速度加快到10倍。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2倍，还假设FP指令占整个执行时间的50%。请比较两种设计方案，决定那种设计对系统的性能影响较大？

解： $S_{FPSQR}=1/((1-0.2)+0.2/10)=1/0.82=1.22$

$$S_{FP}=1/((1-0.5)+0.5/2)=1/0.75=1.33$$

可见第二种方案更好。

Amdahl定律说明：

- I. 某种改进措施可以使整个系统的性能提高多少
- II. 为了改进性能价格比，我们如何合理分配系统的资源

2) 计算机系统设计的基本方法

① 软硬件取舍的基本原则

- 在现有硬件和器件条件下，系统要有尽可能高的性价比，要从实现费用、速度和其他性能要求来综合平衡。
- 系统结构和组成的设计尽可能不要过多或不合理地限制各种组成、实现技术的采用。
- 既要从“硬”的角度考虑如何发挥应用组成和器件技术，又要从“软”的角度考虑如何为编译、操作系统的实现、高级语言程序设计提供更多更好的硬件支持。

② 计算机系统设计者的主要任务

○ 要满足用户对功能、价格和性能的要求。这些要求包括：

- **应用领域需求**：如需专用还是通用？用于科学计算还是普通商用？
- **软件兼容层次需求**：如只需高级语言级兼容，还是需二进制代码级兼容？
- **操作系统需求**：如OS对地址空间大小、存储管理、保护、环境切换、中断/自陷等方面的要求。
- **标准需求**：如浮点标准、I/O总线标准、网络标准等。

○ 在满足功能需求的基础上，进行设计的优化

- 以性价比作为衡量标准，仔细考虑各个功能应该由硬件还是软件实现，在做选择时，必须考虑**设计的复杂性和实现的难易程度**。

○ 设计应能适应日后发展趋势

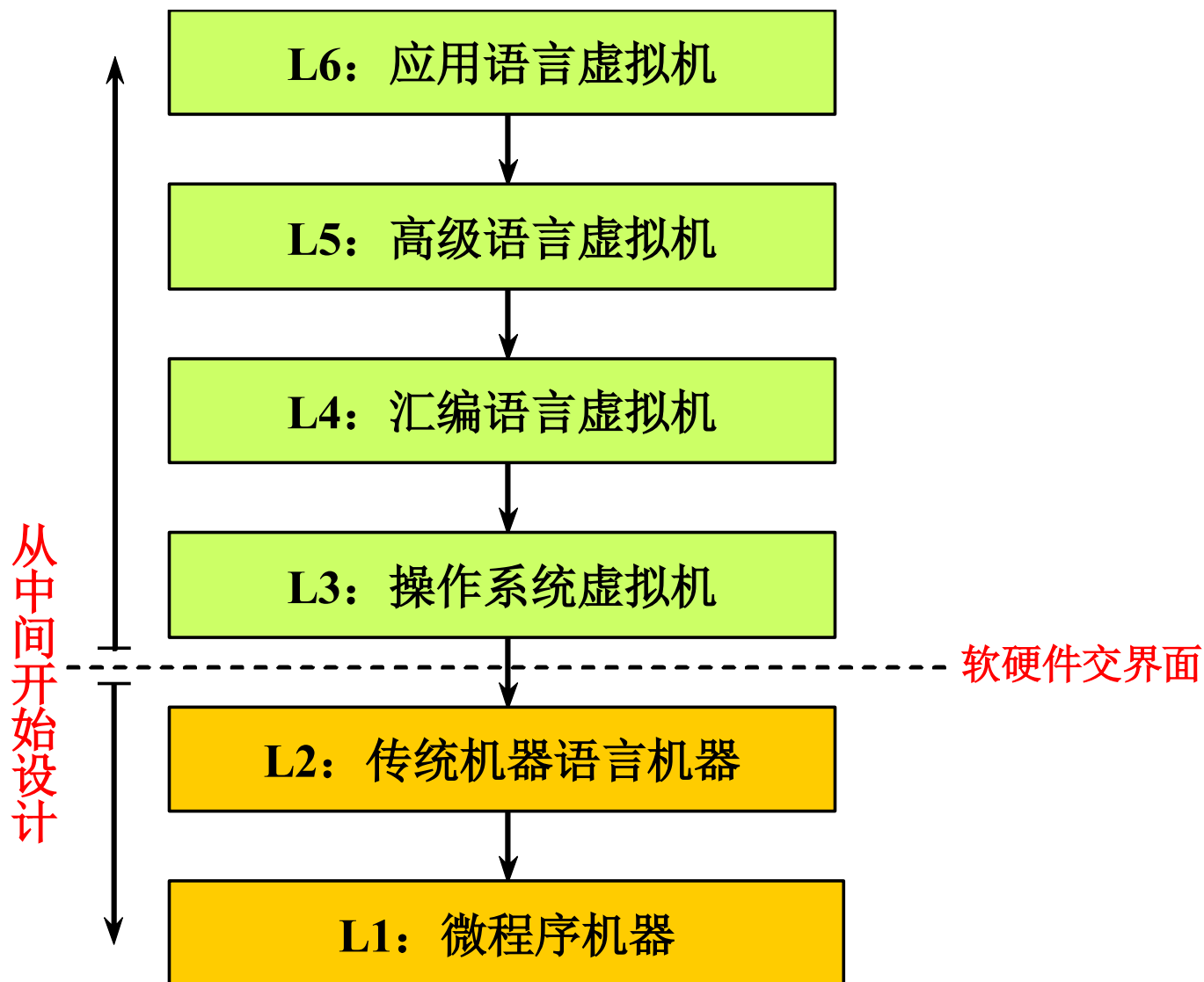
- 好的系统结构设计应**能够经受硬件、软件技术的变革以及用户需求的变化**。

③ 计算机系统设计的基本方法

- “由上往下” (top-down)设计：先考虑如何满足需求，确定使用者那一级机器的基本功能和特性，如基本命令、语言/指令、数据类型和结构等，然后逐级往下设计。此方法适合专用机的设计。
- “由下往上” (bottom-up)设计：不管应用需求，只根据器件，参照已有各种机器的特点，设计出微程序机器级和传统机器级，然后再为不同应用配不同操作系统和编译系统软件，使应用人员根据所提供的语言和数据形式，采用适当算法满足相应的应用。该方法适用于通用机的设计。

③ 计算机系统设计的基本方法

- “由上往下”和“由下往上”设计方法的主要缺点：软、硬件设计分离和脱节。
- “由中间开始”（middle-out）设计（最合理）：
即考虑现有的硬件及器件，又要考虑应用所需的算法和数据结构，先定义好软、硬件的交界面。确定好哪些功能由硬件实现，哪些功能由软件实现，同时还要考虑好硬件对操作系统、编译系统的实现应提供什么样的支持。然后由这个中间点分别往上、往下进行软件和硬件的设计。由于软件和硬件并行设计，可缩短开发周期，而且软硬件设计人员可及时交流协调，所以此方法是目前最常用的、较好的设计方法。



1.5 计算机系统的性能评价

- 1) CPU性能公式
- 2) MIPS和MFLOPS
- 3) 基准测试程序
- 4) 性能评价结果的统计和比较

1) CPU性能公式

- 衡量计算机系统性能好坏，最简单，也是最关键的指标就是**程序在机器上运行所用的CPU时间**。可用以下两种方式表示：
- **CPU时间**= CPU时钟周期数 × 时钟周期长
或
- **CPU时间**= CPU时钟周期数/运行频率
- 其中时钟周期（Clock Cycle）与运行频率（MHz为单位）互为倒数

CPU性能三要素

- 若程序运行时执行的指令条数为 I_N 。
则执行每条指令的平均时钟周期数CPI为：

$$CPI = \text{CPU时钟周期数} / I_N$$

- 由此可得CPU性能公式：

$$T_{\text{CPU}} = I_N \times CPI \times T_C$$

$$T_{\text{CPU}} = I_N \times CPI / R_C$$

- 上式表明：CPU的性能取决于三个要素：

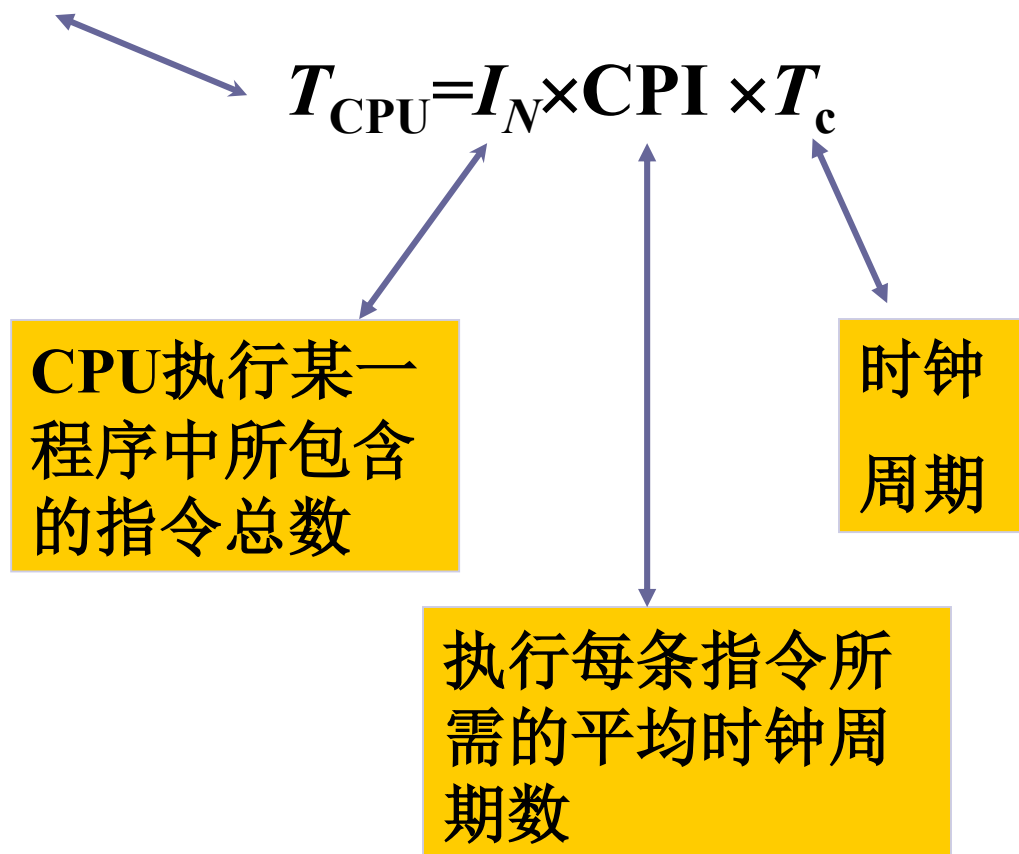
- 时钟频率 R_C ：取决于计算机实现技术和计算机组成；
- 每条指令的平均运行时钟周期数CPI：取决于计算机指令集的设计与实现；
- 指令条数 I_N ：取决于系统结构的指令集和编译技术。

平均CPI的计算

- 实际上，各类指令所需的时钟周期数各不相同，若：
 - I_i 为第 i 类指令在程序中执行的次数
 - CPI_i 为执行第 i 类指令所需的平均时钟周期数
 - 则所有指令的平均CPI计算公式为：

$$CPI = \frac{\sum_{i=1}^n (CPI_i \times I_i)}{\sum_{i=1}^n I_i} = \sum_{i=1}^n (CPI_i \times \frac{I_i}{I_N})$$

CPU性能：CPU执行程序所用的时间。



$$\sum_{i=1}^n (\text{CPI}_i \times I_i)$$

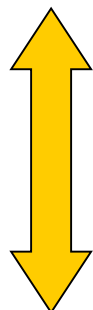
i : 第 i 类指令

I_i : 第 i 类指令的数目

CPI_i : 执行每条 i 类指令所需的周期数

执行整个程序所需的CPU时钟周期数

$$\text{CPI} = \frac{\text{执行整个程序所需的CPU时钟周期数}}{\text{程序中指令的总数}}$$



$$\sum_{i=1}^n (\text{CPI}_i \times I_i)$$

$$\text{CPI} = \frac{\sum_{i=1}^n (\text{CPI}_i \times I_i)}{I_N}$$

第 i 类指令在总程序中占的比例

$$\frac{I_i}{I_N}$$

$$= \sum_{i=1}^n (\text{CPI}_i \times \frac{I_i}{I_N})$$

例：如果FP操作的比例为25%，FP操作的平均CPI=4.0，其他指令的平均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减为2。试利用CPU性能公式比较这两种设计方案哪一个更好？

解：没有采用改进措施之前，原系统的CPI为：

$$\begin{aligned} CPI_{\text{原系统}} &= \sum_{i=1}^n (CPI_i \times \frac{I_i}{I_N}) \\ &= (4 \times 25\%) + (1.33 \times 75\%) = 2.0 \end{aligned}$$

如果FP操作的比例为25%,FP操作的平均CPI=4.0,其他指令的平均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减为2。

采用方案1 (使FPSQR操作的CPI为2) 后, 整个系统的CPI

$$\begin{aligned} \text{CPI}_{\text{方案1}} &= \text{CPI}_{\text{原系统}} - 2\% \times (\text{CPI}_{\text{老FPSQR}} - \text{CPI}_{\text{新FPSQR}}) \\ &= 2.0 - 2\% \times (20 - 2) = 1.64 \end{aligned}$$

采用方案2 (提高所有FP指令处理速度) 后, 整个系统的CPI为:

$$\begin{aligned} \text{CPI}_{\text{方案2}} &= \text{CPI}_{\text{原系统}} - 25\% \times (\text{CPI}_{\text{老FP}} - \text{CPI}_{\text{新FP}}) \\ &= 2.0 - 25\% \times (4 - 2) = 1.5 \end{aligned}$$

显然, 从降低整个系统的指令平均时钟周期数的程度来看, 第二种方案要好些。

方案2的加速比为:

$$\begin{aligned} \text{加速比}_{\text{方案2}} &= \text{CPU时间}_{\text{原系统}} / \text{CPU时间}_{\text{方案2}} \\ &= I_N \times \text{时钟周期} \times \text{CPI}_{\text{原系统}} / I_N \times \text{时钟周期} \times \text{CPI}_{\text{方案2}} \\ &= 2 / 1.5 = 1.33 \end{aligned}$$

2) MIPS、MFLOPS

- 除了CPU时间以外，MIPS、MFLOPS也是比较常用的机器性能的评估标准。
 - MIPS 每秒百万条指令数
 - MFLOPS 每秒百万条浮点操作次数
- 应用范围：
 - MIPS适用于评估标量机
 - MFLOPS适用于评估向量机

MIPS、MFLOPS的计算方法

$$MIPS = \frac{\text{程序指令条数}}{\text{程序执行时间} \times 10^6} = \frac{I_N}{T_E \times 10^6} = \frac{I_N}{I_N \times CPI \times T_C \times 10^6} = \frac{R_C}{CPI \times 10^6}$$

其中 I_N 表示程序中包含的指令条数， T_E 表示程序的执行时间， R_C 为时钟频率，它是时钟周期时间 T_C 的倒数

$$MFLOPS = \frac{\text{程序浮点运算次数}}{\text{程序执行时间} \times 10^6} = \frac{I_{FN}}{T_E \times 10^6}$$

其中 I_{FN} 表示程序中的浮点运算次数

3) 基准测试程序 (Benchmark)

- 为了对计算机性能进行客观评价，常选取一些具有代表性的工作负荷—**基准测试程序 (Benchmark)** 来评价系统性能。常用的基准测试程序有：
 - **实际应用程序**：随应用的不同而变化；
 - **核心程序**：由实际程序中提取的**少量关键循环代码构成的程序**，可以根据机器的使用功能**对某方面性能的要求**来选择核心程序。运行不同的核心程序机器性能可能会有很大不同。
 - **合成测试程序**：也称为测试程序组件，由**一组有代表性的测试程序组成的一个通用的测试程序集**。这是目前日渐普及的测试程序产生方法，它避免了独立测试程序的片面性。

3) 基准测试程序 (Benchmark)

- 为了对计算机性能进行客观评价，常选取一些具有代表性的工作负荷—**基准测试程序 (Benchmark)** 来评价系统性能。常用的基准测试程序有：
 - **小测试程序**：简单的只有几十行的小程序。
 - **核心测试程序**：由实际程序中提取的**少量关键循环代码构成的程序**，可以根据机器的使用功能**对某方面性能**的要求来选择核心程序。运行不同的核心程序机器性能可能会有很大不同。
 - **合成测试程序**：人工合成出来的程序。Whetstone与Dhrystone是最流行的合成测试程序。
 - **基准测试程序套件**：由各种不同的真实应用程序构成。如：SPEC系列（美国的标准性能测试公司创建），**SPEC CPU2006**：29个程序。

4) 性能评价结果的统计和比较

○ 选择工作负载

- 基准测试程序套件
- 核心程序
- 合成测试程序

○ 测试运行

○ 统计比较



	机器A	机器B	机器C	W (1)	W (2)	W (3)
程序1	1.00	10.00	20.00	0.50	0.909	0.999
程序2	1000.00	10.00	20.00	0.50	0.091	0.001
加权算术 平均值 A_m (1)	500.50	10.00	20.00			
加权算术 平均值 A_m (2)	91.91	10.00	20.00			
加权算术 平均值 A_m (3)	2.00	10.00	20.00			

- 选择的测试程序、试验环境不同，结果也有很大区别。
- 如程序1在A机上运行的更快，而程序2却在B机上运行的更快。到底那台机器好？怎样进行综合评测呢？

统计结果

- **总执行时间：** 机器执行所有测试程序的总时间

- B机执行程序1和程序2的速度是A机的50.05倍

- C机执行程序1和程序2的速度是A机的24.02倍

- B机执行程序1和程序2的速度是C机的2倍

- **平均执行时间：** 各测试程序执行时间的**算术平均值**

$$S_m = \frac{1}{n} \sum_{i=1}^n T_i$$

其中： T_i ： 第*i*个测试程序的执行时间

n ： 测试程序组中程序的个数

○ 加权执行时间：各测试程序执行时间的加权平均值

$$A_m = \sum_{i=1}^n W_i \cdot T_i$$

其中， W_i ：第*i*个测试程序在测试程序组中所占的比重

$$\sum_{i=1}^n W_i = 1$$

T_i ：该程序的执行时间

○ 调和平均值法

$$H_m = \frac{n}{\sum_{i=1}^n \frac{1}{R_i}} = \frac{n}{\sum_{i=1}^n T_i}$$

其中， R_i ：由n个程序组成的工作负荷中执行第i个程序的速度

$$R_i = 1/T_i$$

T_i ：第i个程序的执行时间

○ 加权调和平均值公式

$$H_m = \left(\sum_{i=1}^n \frac{W_i}{R_i} \right)^{-1} = \left(\sum_{i=1}^n W_i T_i \right)^{-1}$$

- **几何平均值法**：以某台计算机的性能作为参考标准，其他计算机性能则除以该参考标准而获得一个比值。

$$G_m = \sqrt[n]{\prod_{i=1}^n R_i} = \sqrt[n]{\prod_{i=1}^n \frac{1}{T_i}}$$

R_i ：由n个程序组成的工作负荷中执行第i个程序的速度

$$R_i = 1/T_i$$

\prod ：连乘

○加权几何平均值

$$G_m = \prod_{i=1}^n (R_i)^{W_i} = (R_1)^{W_1} \times (R_2)^{W_2} \times \cdots \times (R_n)^{W_n}$$

○ G_m 表示法有一个很好的特性

几何平均值的比等于比的几何平均值

$$\frac{G_m(x_i)}{G_m(y_i)} = G_m\left(\frac{x_i}{y_i}\right)$$

- 在对各种机器性能比较进行性能规格化过程中，**不论取哪一台机器作参考机**，几何平均 G_m 均能保持比较结果的一致性，而 A_m 和 H_m 都没有这样的特性。
- 若以时间作为惟一衡量标准，用调和平均 H_m 比较精确。
- 若是以某一台机器作为参考机，用几何平均 G_m 比较精确。

例：设有计算机 A, B, C，运行程序 1 和 2。

若以 A 机为参考机，规格化测试结果如下：

测试程序	A 机	B 机	C 机
B1	20(1.00)	10(0.50)	40(2.00)
B2	40(1.00)	80(2.00)	20(0.50)
算术平均 A_m	1.00	1.25	1.25
几何平均 G_m	1.00	1.00	1.00

若以 B 机为参考机，规格化结果如下：

测试程序	A 机	B 机	C 机
B1	20(2.00)	10(1.00)	40(4.00)
B2	40(0.50)	80(1.00)	20(0.25)
算术平均 A_m	1.25	1.00	2.13
几何平均 G_m	1.00	1.00	1.00


因此，在有参考机的情况下选择算术平均作为评价方法是不合适的。
此时，应该选择几何平均。

1.6 计算机系统结构的发展

- 计算机技术的飞速发展得益于两个方面
 - 计算机制造技术的发展
 - 计算机系统结构的创新（I/O工作方式，存储组织，指令系统，并行处理技术）
- 其他因素（如软件、应用）也有很大影响

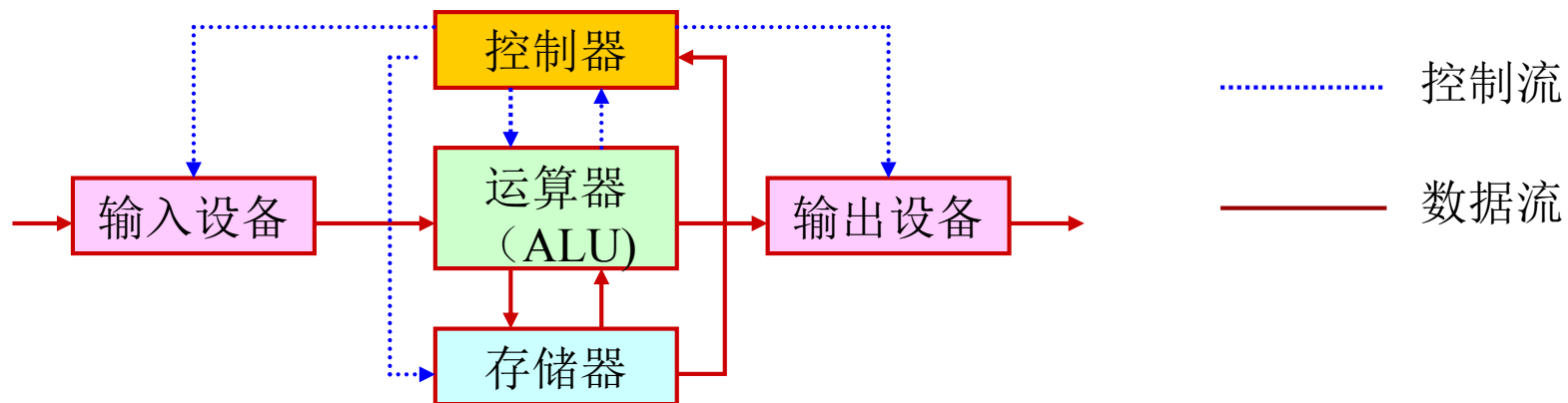
计算机经历的几个发展阶段

阶段	主导因素	年性能增长率
1946年起的25年	两种因素都起着重要的作用	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，制造技术迅猛发展	约35%
20世纪80年代中开始	RISC结构出现，系统结构主导更新和变革，制造技术也不断发展	50%以上
2002年至今	制造技术：功耗问题、存储器访问速度的提高缓慢。 系统结构：可以进一步有效开发的指令级并行性已经很少。	约20%

- 
- 1) 计算机系统结构的演变
 - 2) 软件、应用、器件对计算机系统设计的影响

1) 计算机系统结构的演变

- 第一台通用电子计算机诞生于1946年
- 1946年，冯.诺依曼提出一个完整的现代计算机雏形，包括**运算器**、**控制器**、**存储器**和**输入输出设备**等。
- 冯.诺依曼结构**



冯.诺依曼结构计算机的主要特征

- 以运算器为中心，I/O设备与存储器之间的数据传送都途经运算器。
- 运算器、存储器、I/O设备的操作以及它们之间的联系都由控制器集中控制。
- 存储器按一维线性编址，顺序访问存储器地址单元，每个存储单元的位数固定，地址唯一。程序存储，指令和数据都存放在存储器中。
- 由指令形式的低级机器语言驱动。指令顺序执行，由一个顺序控制器指定即将被执行的指令地址。指令由操作码和地址码组成。
- 数据以二进制表示。

对冯.诺依曼结构计算机的改良

- 增加了新的数据表示，如浮点、字符串、十进制等
- 采用虚拟存储器，方便了高级语言编程
- 引入堆栈，支持过程调用、递归等
- 采用变址寄存器，并增加了间接寻址方式
- 增加CPU中寄存器数量，引入Cache，避免频繁访存
- 采用存储器交叉访问技术，以及无冲突并行存储器，增加了存储器带宽
- 采用了流水技术，以加快指令及操作的执行速度
- 采用多功能部件。从而一条指令可对多个数据元素在不同功能部件上并发操作
- 采用支持处理机，如协处理机、I/O处理机，保证CPU专注于数值运算
- 采用自定义数据表示，由数据中的标志符显示说明自身属性
- 使程序和数据空间分开，从而增加了存储器带宽
- 这些改进措施使计算机系统结构已由原来的以运算器为中心演变为以存储器为中心

近40年计算机系统结构的重大改进

- 1.计算机系统结构从串行到并行，出现了向量计算机、并行计算机、多处理机等。
- 2.出现了面向高级语言机器和直接执行高级语言机器。
- 3.出现了面向操作系统机器和数据库计算机。
- 4.出现了数据流计算机和归约机。
- 5.出现了各种专用计算机，容错计算机。
- 6.出现了各种功能分布计算机、通信处理机、大规模和超大规模计算机、智能计算机。
- 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。
- 主要方向：智能化，高性能，专业化等。

2) 软件、应用、器件对系统设计发展的影响

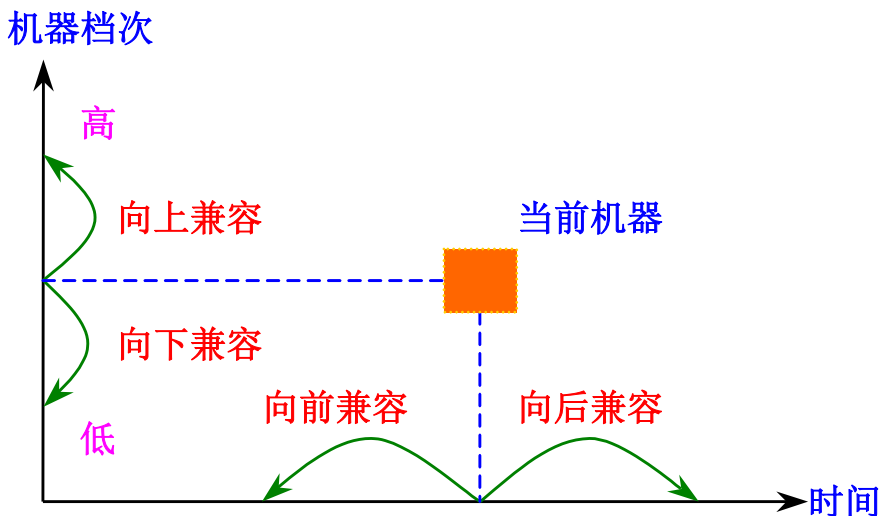
○ 软件对系统结构的影响

- 软件是促使计算机系统结构发展的最重要因素。
- **可移植性**：是指一个软件可不经修改或只需少量修改便可由一台机器移植到另一台机器上去运行，即同一软件可应用于不同的环境。

如何解决软件的可移植性问题？

○ 1.采用系列机方法

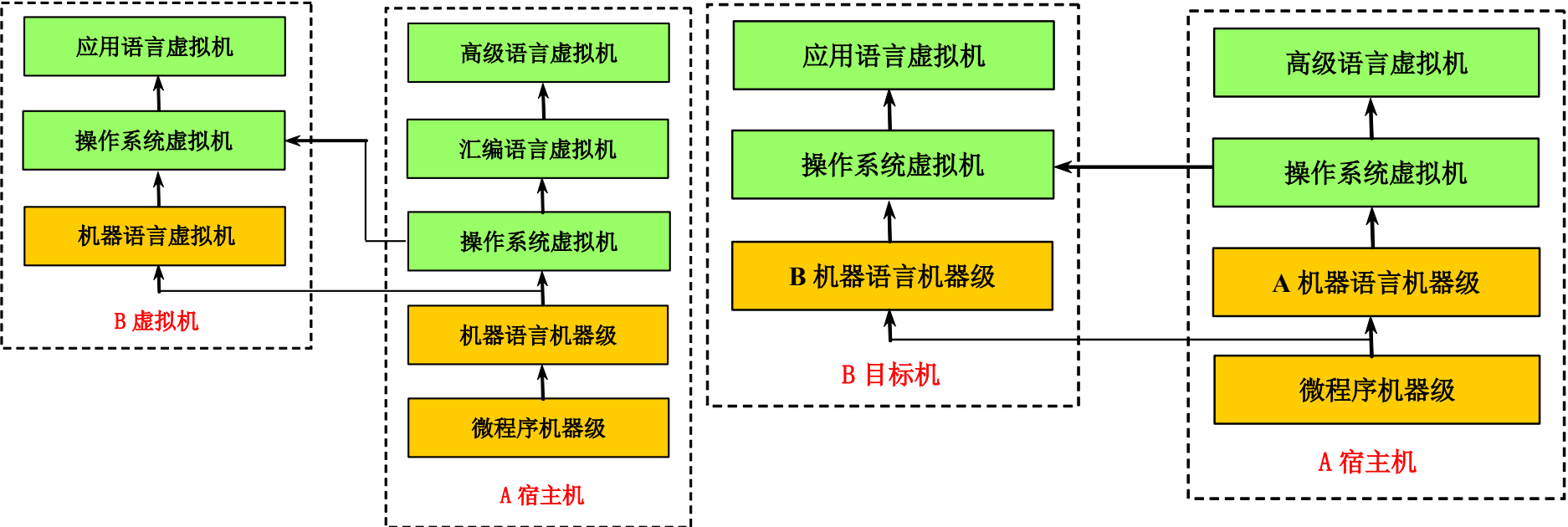
- **系列机（兼容机）**：同一厂家生产的具有相同的系统结构，但具有不同的组成和实现的一系列不同型号的机器。
- **向上兼容**：为某档机种编制的软件应能不加修改地运行于比它高的档次的机种上；
- **向后兼容**：为某个时期投入市场的机种编制的软件应能不加修改地运行于在它之后投入市场的机种上。同理，有向下兼容和向前兼容。**兼容机**：由不同公司厂家生产的具有相同系统结构的计算机。



如何解决软件的可移植性问题？

○ 2.采用模拟和仿真方法

- 使软件能在具有不同系统结构的机器之间相互移植。在一种系统结构上实现另一种系统结构。从指令集的角度来看，就是要在一种机器上实现另一种机器的指令集。
- **模拟**：指用软件的方法在一台计算机(**宿主机A**)上，实现另一台计算机(**虚拟机B**)的指令系统，由于用纯软件解释执行，速度较慢。
- **仿真**：用微程序的方法在一台计算机(**宿主机A**)上实现另一台计算机(**目标机B**)的指令系统，由于有部分硬件/固件参与解释执行，故速度比“模拟”快。



模拟

仿真

如何解决软件的可移植性问题？

○ 3.采用统一标准高级语言方法

- 实现软件移植的一种理想的方法
- 较难实现

○ 应用对系统结构的影响

- 计算机应用是促使计算机系统结构发展的最根本动力，不断推动计算机系统结构朝着高运算速度、大存储容量和大I/O吞吐率三个方面发展。

○ 器件对系统结构的影响

- 器件是促使计算机系统结构发展的最活跃因素，器件的性能价格比不断提高，特别是LSI和VLSI器件的发展，使计算机的价格发生了重大变化。
- 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。

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

○ 算法和系统结构

- 1. 系统结构设计本身即是算法的设计，又是系统结构的设计。需要结合两者的优势。
- 2. 需要不断改进基本的系统结构。
- 3. 需要充分利用并行性，获得高速度。

○ 综上所述，计算机系统结构是把各种功能部件组成一个系统，这些部件可以是硬件、软件或两者的混合体。系统结构设计是选择一种最佳的部件组合，使得整个系统能更有效的工作。

1.7 并行系统结构概念

- **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。

1.7 并行系统结构概念

- 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/程序。

1.7 并行系统结构概念

○ 提高并行性的技术途径：

○ 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。

○ 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。

○ 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。

1.8 小结

- 本章介绍计算机系统结构的一些基本知识。包括定性知识和定量知识两部分内容。
 - **定性知识**：本课程经常使用的一些名词概念，以及对计算机的定性认识、分析方法。如计算机系统的多层次结构，计算机结构、组成和实现，软件移植性，兼容性，透明性，计算机系统的分类方法等。
 - **定量知识**：对计算机性能进行定量评价的基本原则以及几个重要公式。如CPU性能公式、Amdahl定律、MIPS、MFLOPS计算公式等。

1.9 习题

○ 名词解释举例

- 计算机系统结构
- 翻译 解释 透明性
- 系列机
- Amdahl定律 Flynn分类法 程序访问局部性
- CPI, MIPS, MFLOPS

○ 简答题举例

- 计算机系统的Flynn分类法是按照什么来分类的？共分为哪几类？简要说明各类的特征。
- 简述实现软件移植的三个途径？
- 计算机系统设计应遵循什么定量原理？
- 简述计算机系统设计中软硬件取舍的三个原则。

名词解释

1. 系列机

【答案】

所谓系列机是指在一个厂家内生产的具有相同的系统结构，但具有不同组成和实现的一系列不同型号的机器。系列机方法能够在具有相同系统结构的各种机器之间实现软件移植。

2、计算机系统结构,计算机组成,计算机实现

【答案】

计算机系统结构、计算机组成和计算机实现是三个不同的概念。计算机系统结构是指机器语言或汇编语言程序员所看到的硬件子系统的概念性结构和功能特性，它是计算机系统软、硬件的交界面。计算机组成是计算机系统结构的逻辑实现，包括机器内部的数据流和控制流的组成以及逻辑设计等。计算机实现是指计算机组成的物理实现。

选择题

1. SISD是指（）

- A.单指令流单数据流 B.单指令流多数据流
C.多指令流单数据流 D.多指令流多数据流

【答案】 A

2.对计算机系统结构，下列（）是透明的。

- A.浮点数据表示 B.指令系统
C.访问方式保护 D.阵列运算部件

【答案】 D

3.下列（）兼容方式对系列机来说是必须做到的。

- A.向前兼容 B.向后兼容
C.向上兼容 D.向下兼容

【答案】 B

在某台计算机上, 根据程序跟踪实验结果, 已知每种指令所占的比例及CPI如下:

求: 上述情况的平均CPI。

指令类型	指令所占比例	CPI
算逻指令	43%	1
LOAD指令	21%	2
SRORE指令	12%	2
转移指令	24%	2

$$\begin{aligned}\text{CPI} &= 1 \times 0.43 + 2 \times 0.21 + 2 \times 0.12 + 2 \times 0.24 \\ &= 0.43 + 0.42 + 0.24 + 0.48 = 1.57\end{aligned}$$

- 4.用一台40MHz处理机执行标准测试程序，它所包含的各种指令的时钟周期数及指令条数如下表所示：

指令类型	指令数	时钟周期数
整数运算	45,000	1
数据传送	32,000	2
浮点	15,000	2
控制传送	8,000	2

求平均CPI、MIPS速率和程序的执行时间。

- 答：由各种指令条数可以得到总条数，以及各类指令的百分比，然后代入**CPU性能公式**计算：

$$\text{全部指令条数: } I_N = \sum_{i=1}^4 I_i = 100000 = 10^5$$

指令的平均运行时钟周期数：

$$CPI = \sum_{i=1}^4 (CPI_i \times \frac{I_i}{I_N}) = 1 \times 0.45 + 2 \times 0.32 + 2 \times 0.15 + 2 \times 0.08 = 1.55$$

$$MIPS = \frac{R_C}{CPI \times 10^6} = \frac{40 \times 10^6}{1.55 \times 10^6} = \frac{40}{1.55} \approx 25.806$$

整个程序的执行时间：

$$T_E = \frac{I_N}{MIPS \times 10^6} = \frac{10^5}{25.806 \times 10^6} \approx 0.003875 \text{ (秒) 或:}$$

$$T_E = I_N \times CPI \times T_C = I_N \times CPI \times \frac{1}{R_C} = 10^5 \times 1.55 \times \frac{1}{40 \times 10^6} = 0.003875 \text{ (秒)}$$

- 假设高速缓存Cache的工作速度为主存的5倍，且Cache被访问命中的概率为90%，则采用Cache后，能使整个存储系统获得多高的加速比？
- 答：根据**Amdahl定律**，整个存储系统加速比为：

$$S_p = \frac{T_0}{T_e} = \frac{1}{(1 - f_e) + f_e / r_e} = \frac{1}{(1 - 0.9) + 0.9 / 5} = 3.57$$

- 某计算机系统采用图形协处理器后使图形运算速度提高到原来的20倍，而系统运行某一程序的速度提高到原来的5倍，问：该程序中图形运算所占的比例是多少？
- 答：根据Amdahl定律，有

$$5 = \frac{1}{(1 - f_e) + \frac{f_e}{20}} = \frac{1}{1 - \frac{19}{20} f_e}$$

得图形运算所占比例为 $f_e = 0.8421052631 \approx 84.2\%$

- 在一台标量计算机中增加了一个向量运算部件，向量运算的速度为标量运算速度的20倍，程序中可用向量方式求解部分所占的百分比称为可向量化百分比，原系统运算时间与采用向量部件后系统运算时间之比称为系统的加速比。
- (1) 试画出加速比与可向量化百分比两者关系的曲线。
- (2) 为达到加速比2，可向量化百分比应为多少？
- (3) 为获得采用向量方式最大加速比20的一半时，所需的可向量化百分比为多少？
- (4) 如果程序可向量化百分比为70%。如果想继续提高性能10%，是采用硬件方法继续提高向量部件的速度好，还是通过编译程序进一步提高向量化比的方法好？

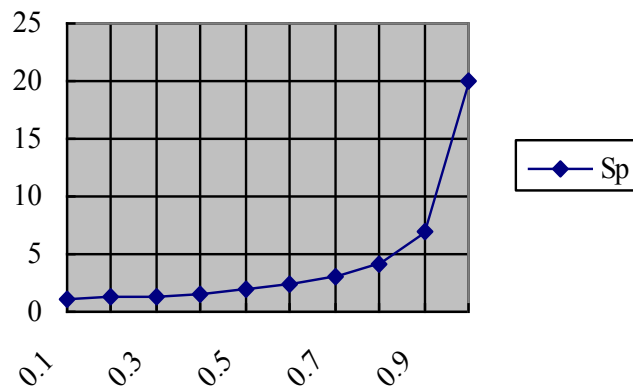
(1) 试画出加速比与可向量化百分比两者关系的曲线。

○ 答：(1) 设可向量化比例为 f_e ，根据Amdahl定律

$$S_p = \frac{1}{(1 - f_e) + \frac{f_e}{r_e}} = \frac{1}{(1 - f_e) + \frac{f_e}{20}} = \frac{1}{1 - \frac{19}{20}f_e}$$

○ 在各 f_e 取值下的 S_p 如下表所示：

f_e	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
S_p	1.10	1.23	1.40	1.61	1.90	2.33	2.99	4.17	6.90	20



(2) 为达到加速比2，可向量化百分比应为多少？

- (2) $S_p=2$ 时，有

$$2 = \frac{1}{1 - \frac{19}{20} f_e}$$

- 故可向量化部分比例为：

$$f_e = \left(1 - \frac{1}{2}\right) \times \frac{20}{19} = \frac{10}{19} = 52.69\%$$

(3) 为获得采用向量方式最大加速比20的一半时，所需的可向量化百分比为多少？

○ (3) $S_p=10$

$$10 = \frac{1}{1 - \frac{19}{20} f_e}$$

$$f_e = \left(1 - \frac{1}{10}\right) \times \frac{20}{19} = \frac{9}{10} \times \frac{20}{19} = \frac{18}{19} = 94.73\%$$

(4) 如果程序可向量化百分比为70%。如果想继续提高性能10%，是采用硬件方法继续提高向量部件的速度好，还是通过编译程序进一步提高向量化比的方法好？

$$(4) f_e=0.7 \text{ 时 } S_p = \frac{1}{1 - \frac{19}{20} \times 0.7} = 2.99$$

提高性能10%后，整个系统的加速比应达到

$$S_p = 2.99 \times 1.1 = 3.29$$

需要向量部件的加速比要达到：

$$3.29 = \frac{1}{1 - 0.7 + \frac{0.7}{r_e}} \quad r_e = 177$$

而通过提高 f_e 的方法，达到同样的加速比

$$3.29 = \frac{1}{1 - f_{e1} + \frac{f_{e1}}{20}} \quad f_{e1} = \left(1 - \frac{1}{3.29}\right) \times \frac{20}{19} = 0.7327 \quad f_{e1} - f_e = 0.7327 - 0.7 = 0.0327$$

即只需提高向量化3.27个百分点，可见采用软件方法较好。

作业

- 1(透明性多级 层次结构, Amdahl定律, 程序局部性原理)
- 3, 5, 7, 9, 10, 11