计算机系统结构

Computer Systems and Architecture

第1章 基本概念

肖梦白
xiaomb@sdu.edu.cn
https://xiaomengbai.github.io

第1章 基本概念

1.1 计算机系统结构简介

1.2 计算机系统的设计技术

1.3 计算机系统的评价标准

1.4 计算机系统结构的发展

1.5 计算机系统结构并行性的发展

1.1 计算机系统结构简介

1.1.1 计算机系统层次结构

1.1.2 计算机系统结构定义

1.1.3 计算机组成和实现

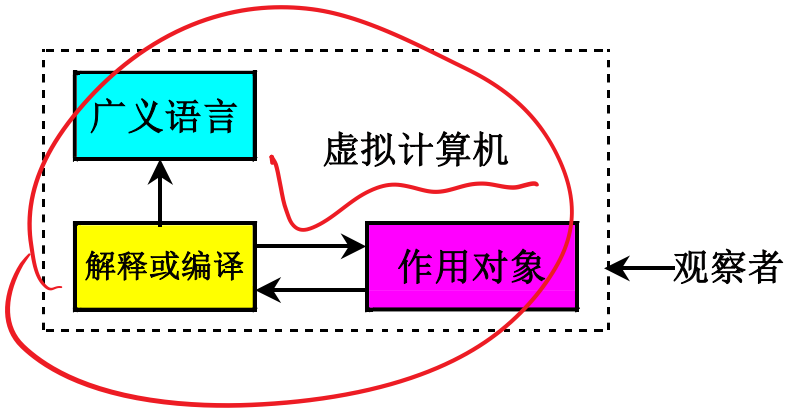
1.1.4 计算机系统结构、组成和实现相互影响

1.1.5 计算机系统结构的分类

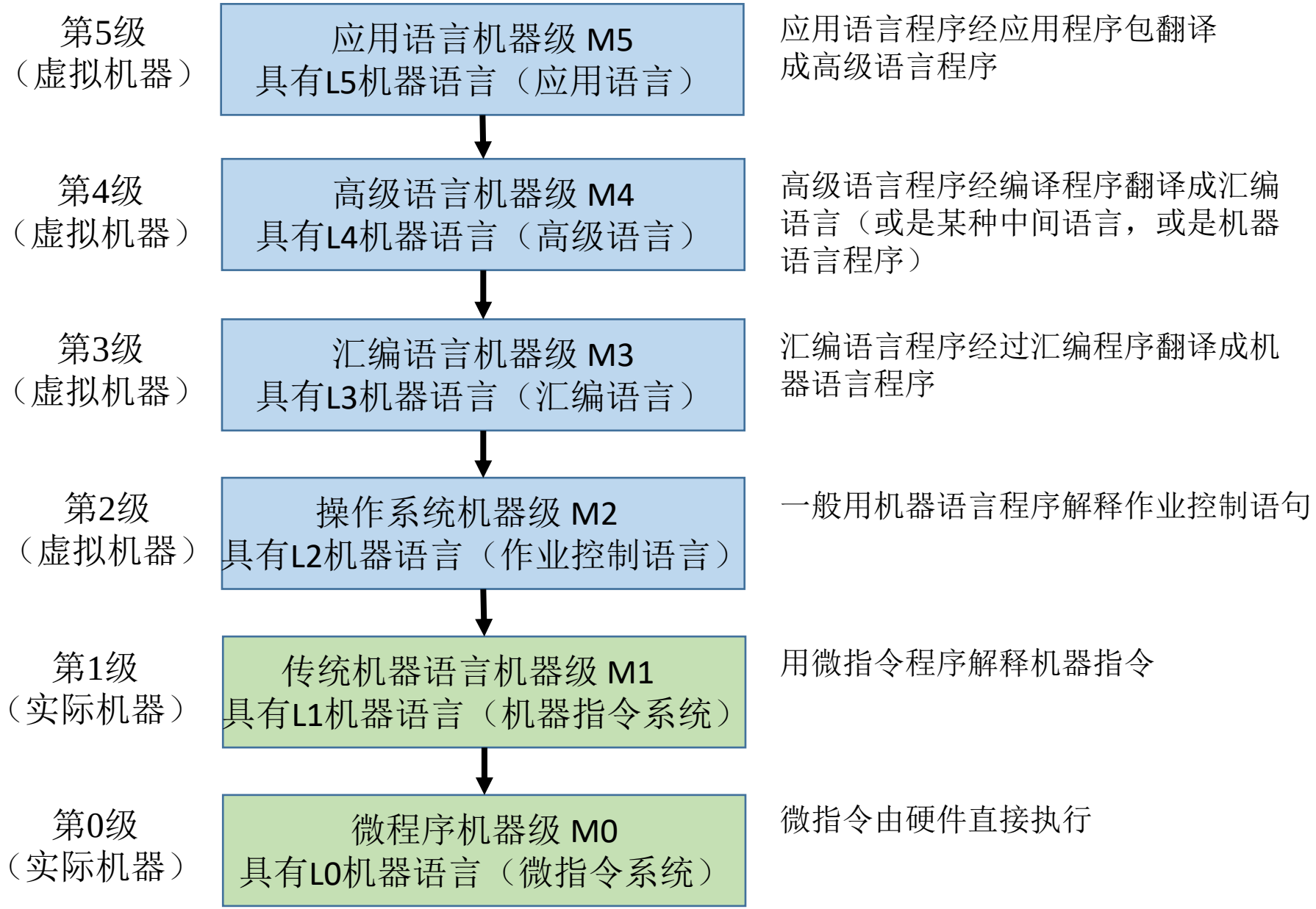
1.1.1 计算机系统层次结构

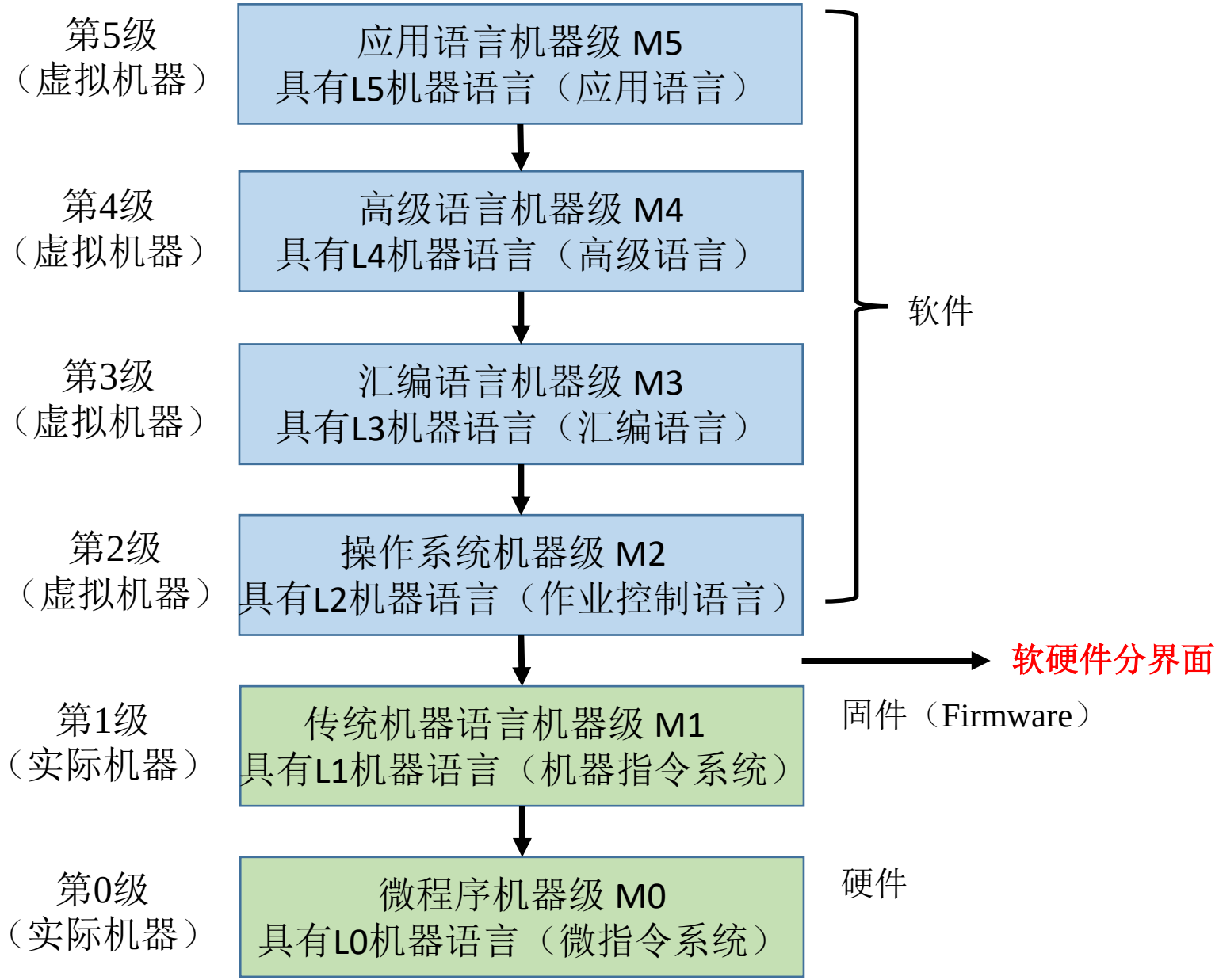
1.虚拟计算机

- 定义：从不同角度所看到的计算机系统的属性是不同的
- 主要观察角度包括：
  - 应用程序员
  - 系统程序员
  - 硬件设计人员



- 对计算机系统的认识通常只需要在某一个层次上


第5级
（虚拟机器）
应用语言机器级 M5
具有L5机器语言（应用语言）
应用语言程序经应用程序包翻译成高级语言程序
第4级
（虚拟机器）
高级语言机器级 M4
具有L4机器语言（高级语言）
高级语言程序经编译程序翻译成汇编语言（或是某种中间语言，或是机器语言程序）
第3级
（虚拟机器）
汇编语言机器级 M3
具有L3机器语言（汇编语言）
汇编语言程序经过汇编程序翻译成机器语言程序
第2级
（虚拟机器）
操作系统机器级 M2
具有L2机器语言（作业控制语言）
一般用机器语言程序解释作业控制语句
第1级
（实际机器）
传统机器语言机器级 M1
具有L1机器语言（机器指令系统）
用微指令程序解释机器指令
第0级
（实际机器）
微程序机器级 M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行


第5级
（虚拟机器）
应用语言机器级 M5
具有L5机器语言（应用语言）
第4级
（虚拟机器）
高级语言机器级 M4
具有L4机器语言（高级语言）
第3级
（虚拟机器）
汇编语言机器级 M3
具有L3机器语言（汇编语言）
第2级
（虚拟机器）
操作系统机器级 M2
具有L2机器语言（作业控制语言）
软件
软硬件分界面
第1级
（实际机器）
传统机器语言机器级 M1
具有L1机器语言（机器指令系统）
固件（Firmware）
第0级
（实际机器）
微程序机器级 M0
具有L0机器语言（微指令系统）
硬件

- **语言实现的两种基本技术：**
  - **翻译**（Translation）：先把N+1级程序全部转换成N级程序后，再去执行新产生的N级程序，在执行过程中N+1级程序不再被访问。
  - **解释**（Interpretation）：每当一条N+1级指令被译码后，就直接去执行一串等效的N级指令，然后再去取下一条N+1级的指令，依此重复进行。
  - 解释执行比编译后再执行所花的时间多，但占用的存储空间较少。
- L0-L2级用解释的方法实现，L3-L5级则多用翻译的方法实现

1.1.2 计算机系统结构的定义

1 定义一

- Amdahl于1964年在推出IBM360系列计算机时提出
- 程序设计者所看到的计算机系统的属性，即概念性结构和功能特性
  - 程序员：系统程序员（包括：汇编语言、机器语言、编译程序、操作系统）
  - 看到的：编写出能在机器上正确运行的程序所必须了解到的
- 按照计算机系统的多级层次结构，不同层级的程序员看到的计算机具有不同的属性；底层计算机的属性对于高层计算机的程序员是**透明的**
  - **透明性：本来存在的事物或属性，从某种角度看似乎不存在**

透明性概念

- 定义：本来存在的事物或属性，从某种角度看似乎不存在
- 例如：CPU类型、型号、主存储器容量等

  对应用程序员　　透明

  对系统程序员、硬件设计人员等　　不透明
- 例如：浮点数表示、乘法指令

  对高级语言程序员、应用程序员　　透明

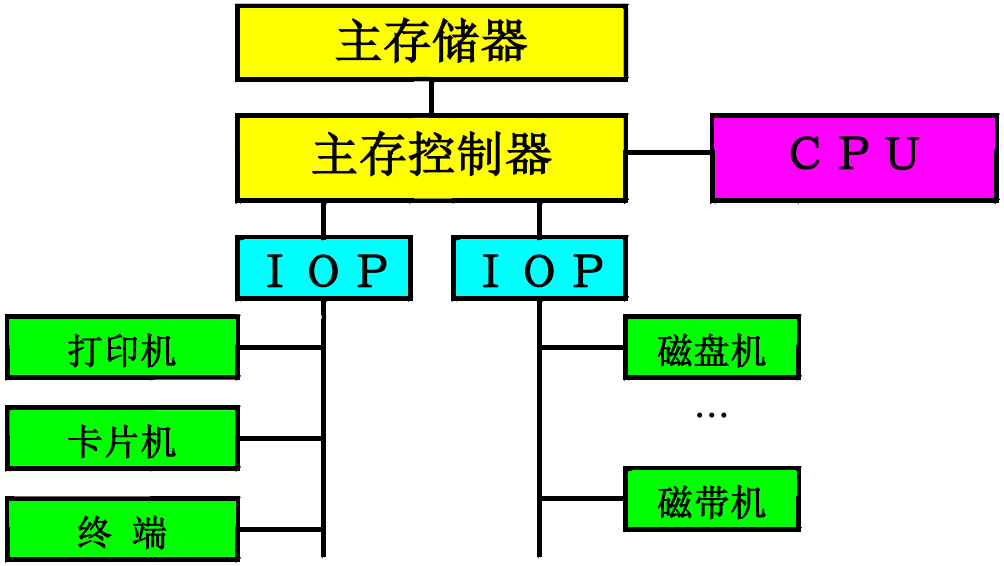
  对汇编语言程序员、机器语言程序员　　不透明
- 例如：数据总线宽度、微程序

  对汇编语言程序员、机器语言程序员　　透明

  对硬件设计人员、计算机维修人员　　不透明

概念性结构

IBM360系列计算机的概念性结构

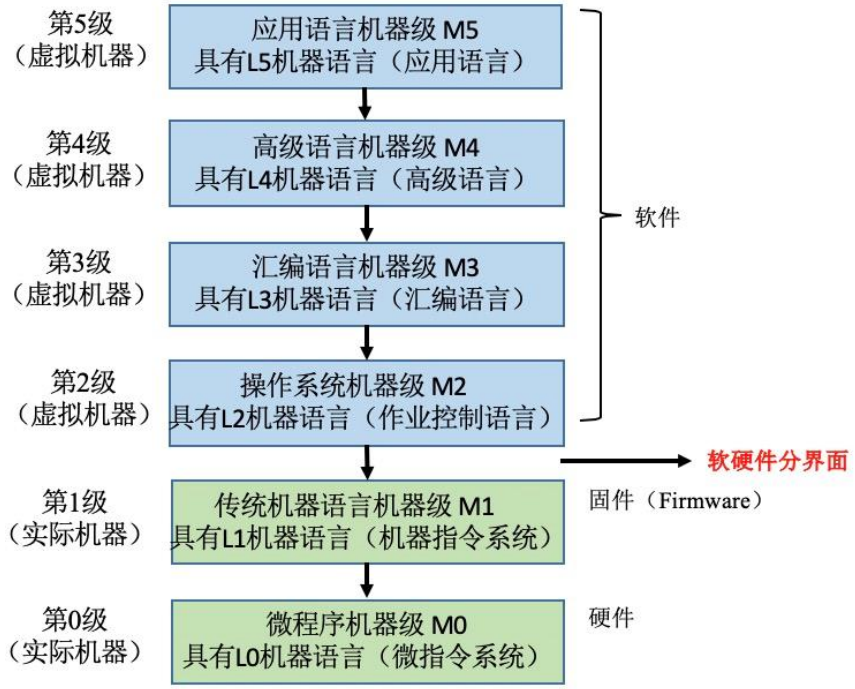


功能特性 指令系统及其执行模式

- **数据表示**：硬件能够直接认别和处理的数据类型；
- **寻址技术**：编址方式、寻址方式和定位方式等；
- **寄存器组织**：操作数寄存器、变址寄存器、控制寄存器及专用寄存器的定义、数量和使用规则等；
- **指令系统**：操作类型、格式，指令间的排序控制等；
- **中断系统**：中断类型、中断级别和中断响应方式等；
- **存储系统**：寻址空间、虚拟存储器、Cache存储器等；
- **处理机工作状态**：定义和切换方式，如管态和目态等；
- **输入输出系统**：数据交换方式、交换过程的控制等；
- **信息保护**：信息保护方式和硬件对信息保护的支持等。

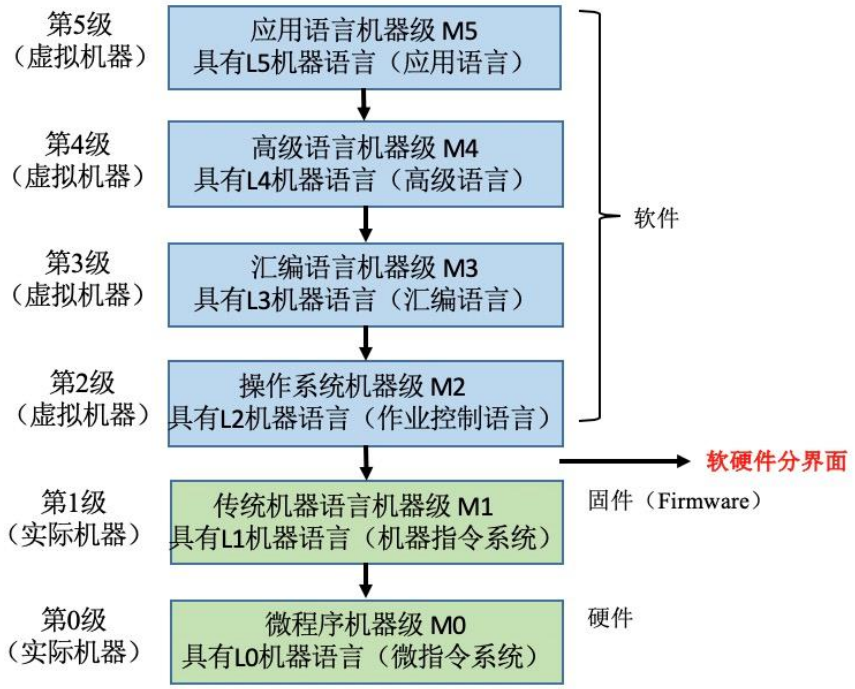
2. 定义二

- 计算机系统结构也称为计算机系统的体系结构（Computer Architecture）
  - 传统机器级的系统结构
  - 软件和硬件/固件的交界面，是机器语言、汇编语言程序设计者，或编译程序设计者看到的机器物理系统的抽象



3. 结论

计算机系统结构研究的是**软、硬件之间的功能分配**以及**对传统机器级界面的确定**，为机器语言、汇编语言程序设计者或编译程序生成系统提供使其设计或生成的程序能在机器上正确运行而应看到和遵循的计算机属性



1.1.3 计算机组成和实现

1.计算机组成（Computer Organization）

- 计算机系统结构的**逻辑实现**，包含物理机器级中的数据流和控制流的组成以及逻辑设计等，着眼于物理机器级内各事件的排序方式与控制方式、各部件的功能以及各部件之间的联系
- 主要研究内容：确定数据通路的宽度；确定各种操作对功能部件的共享程度；确定专用的功能部件；确定功能部件的并行度；设计缓冲和排队策略；设计控制机构；确定采用何种可靠性技术。

1.1.3 计算机组成和实现

2.计算机实现（Computer Implementation）

- 计算机组成的物理实现，
- 主要包括
  - 处理机、主存储器等部件的物理结构；
  - 器件的集成度和速度；
  - 专用器件的设计；
  - 器件、模块、插件、底版的划分与连接；
  - 信号传输技术；
  - 电源、冷却及装配技术，制造工艺及技术等。

- **指令系统的确定**属于计算机系统结构的研究范畴；而**指令的实现**（如取指令、操作码译码、计算操作数地址、取数等操作安排和排序）属于计算机组成的研究范畴；**实现这些指令功能的具体电路、器件的设计以及装配技术**属于计算机实现的研究范畴
- **主存容量与编址方式的确定**属于计算机系统结构的研究范畴；而**主存速度、逻辑结构的设置**属于计算机组成的研究范畴；**主存器件的选定、逻辑设计、微组装技术的使用**属于计算机实现的研究范畴
- 以乘法运算为例，试说明计算机系统结构、计算机组成、计算机实现各考虑什么？

1.1.4 计算机系统结构、组成和实现相互影响

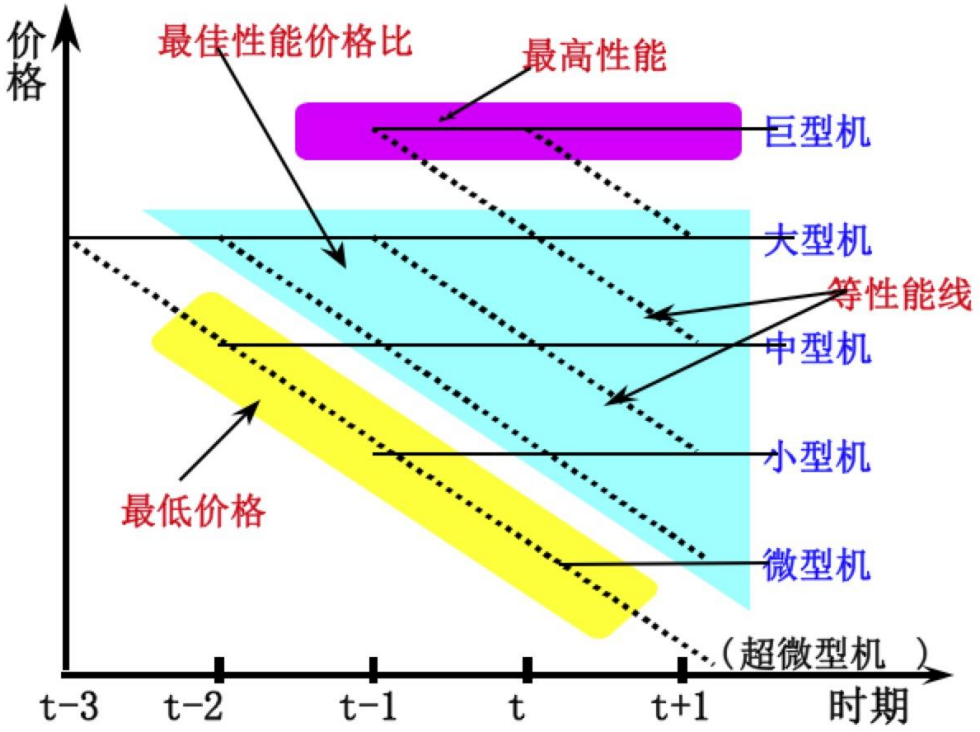
- 三者互不相同，又相互影响
- 相同的系统结构，可以因速度的不同采用不同的组成
  - 指令既可以顺序执行，也可以重叠执行
  - 乘法指令既可以利用专门的乘法器实现，也可以利用加法器和移位器实现，这取决于性能、价格、乘法指令使用频度等
- 一种计算机组成可以有不同的实现方法
  - 主存器件既可以用双极型，也可以用MOS型器件，这取决于性能价格比以及器件技术情况

1.1.5 计算机系统的分类

1. 按处理机性能分类
2. 佛林（Flynn）分类法
3. 库克分类法

1 按处理机性能分类

- 按性能和价格的综合指标划分：巨型、大型、中型、小型、微型机

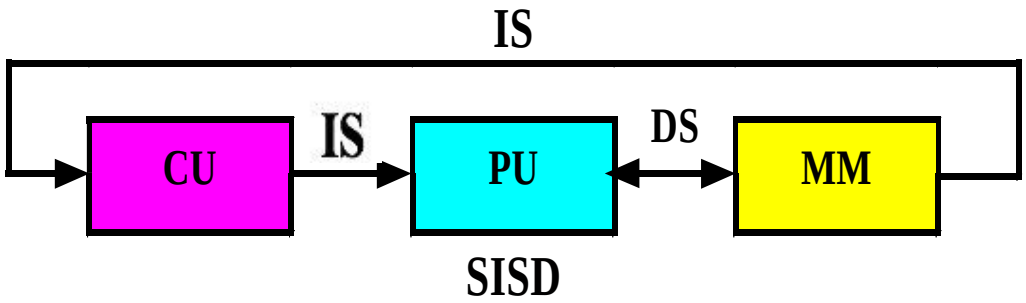


2 佛林（Flynn）分类法

- 1966年由Michael.J. Flynn 提出，按照指令流和数据流的多倍性特征进行分类
  - **指令流（Instruction Stream）：** 机器执行的指令序列
  - **数据流（Data Stream）：** 由指令流调用的数据序列
  - **多倍性(multiplicity)：** 在系统性能瓶颈部件上同时处于同一执行阶段的指令或数据的最大可能个数
- 四种类型
  1) 单指令流单数据流 SISD（Single Instruction Single Data stream）
  2) 单指令流多数据流 SIMD（Single Instruction Multiple Data stream）
  3) 多指令流单数据流 MISD（Multiple Instruction Single Data stream）
  4) 多指令流多数据流 MIMD（Multiple Instruction Multiple Data stream）

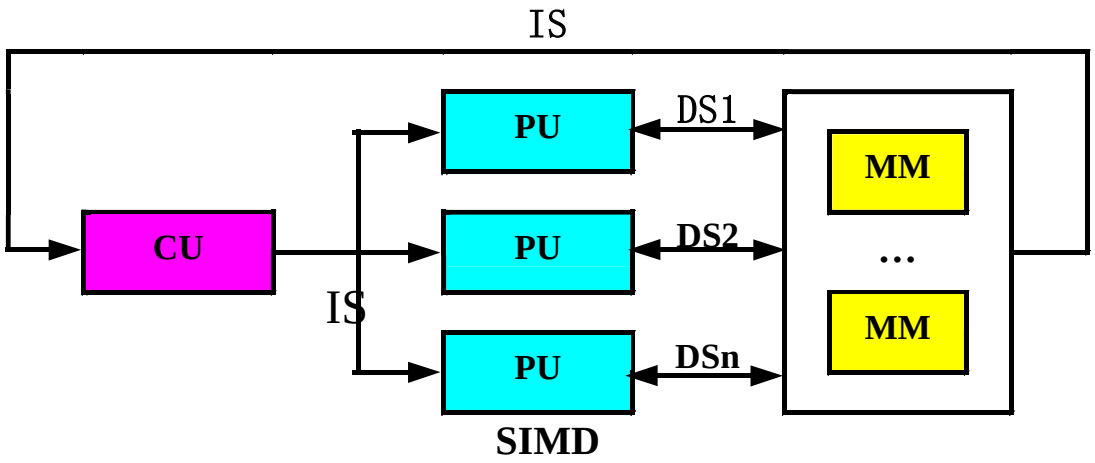
(1) SISD

- 典型顺序处理计算机



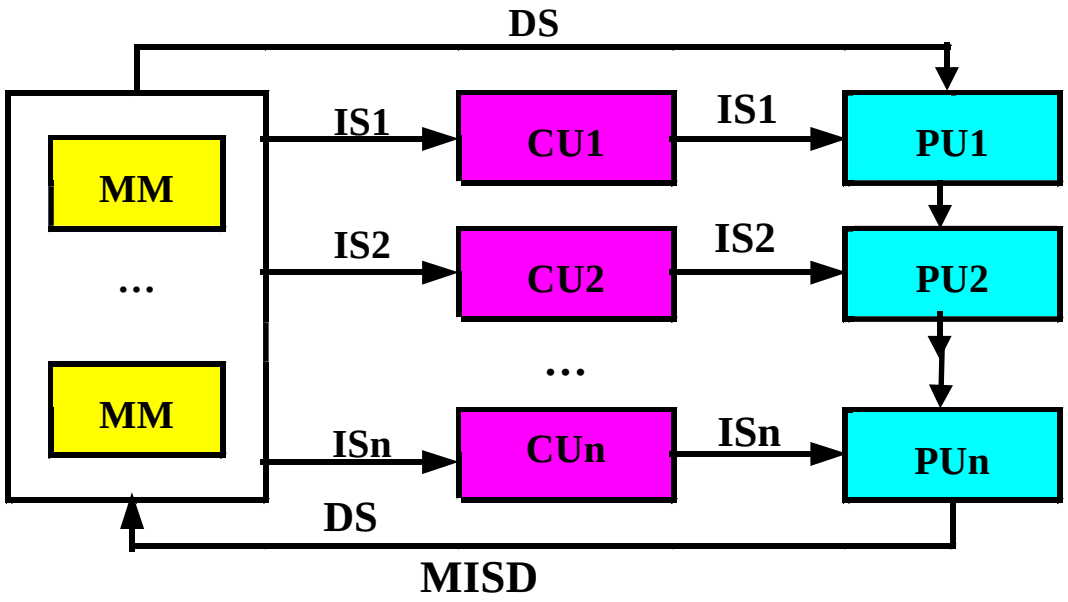
(2) SIMD：

- 并行处理机、阵列处理机、向量处理机、相联处理机、超标量处理机、超流水线处理机
- 多个PU按一定方式互连，在同一个CU控制下，对各自的数据完成同一条指令规定的操作；从CU看指令顺序执行，从PU看数据并行执行。



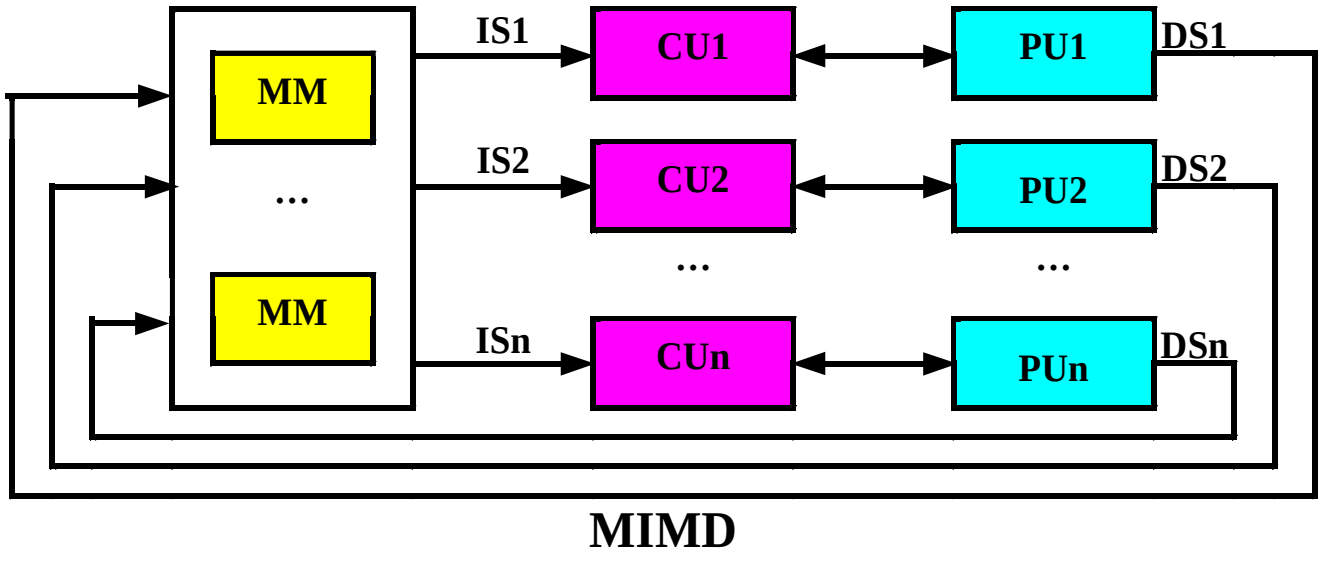
(3) MISD:

几条指令对同一个数据进行不同的处理



(4) MIMD 多处理机系统

- 紧密偶合：IBM3081、IBM3084、UNIVAC-1100/80
- 松散偶合：D-825, Cmmp, CRAY-2



3 库克分类法

- 1978年由 D. J. Kuck提出，按指令流和执行流分类

1） 单指令流单执行流

- SISE(Single Instruction Single Execution stream)
- 典型的单处理机

2） 单指令流多执行流

- SIME(Single Instruction Multiple Execution stream)
- 多功能部件处理机、相联处理机、向量处理机、流水线处理机、超流水线处理机、超标量处理机、SIMD并行处理机

3） 多指令流单执行流

- MISE，(Multiple Instruction Single Execution stream)
- 多道程序系统

4） 多指令流多执行流

- MIME， (Multiple Instruction Multiple Execution stream)
- 典型的多处理机

1.2 计算机系统的设计技术

1.2.1 计算机系统的定量原理

1.2.2 计算机系统设计者的主要任务

1.2.3 计算机系统设计方法

1.2.1 计算机系统的定量原理

1. 哈夫曼（Huffman）压缩原理
2. Amdahl定律
3. CPU性能公式
4. 访问的局部性原理

1. 哈夫曼（Huffman）压缩原理

- 在计算机系统的设计中经常要在不同的方法之间进行折中，尽可能加速处理高概率事件远比加速处理低概率事件对性能的提高要显著
- 例如，CPU在运算中发生溢出的概率是很低的，为此，设计时可考虑加快不溢出时的运算速度，而对溢出时的运算速度不予考虑
- 对经常性时间的优化实现能在多大程度上改进整个系统的性能？

2. Amdahl定律

- 系统中某一部件由于采用更快的执行方式后，整个系统性能的提高与这种执行方式的使用频率或占总执行时间的比例有关

$$\text{可改进部分的比例：}Fe=\frac{\text{可改进部分的执行时间}}{\text{改进前整个任务的执行时间}}$$

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

- 假设T_0为改进前整个任务的执行时间，则改进后整个任务的执行时间为

$$T_n = T_0 \cdot \left(1 - F_e + \frac{F_e}{S_e}\right)$$

- 改进后整个系统的加速比为

$$S_n = \frac{T_0}{T_n} = \frac{1}{1 - F_e + \frac{F_e}{S_e}}$$

例：某部件的处理时间仅为整个运行时间的40%，如果将该部件的处理速度加快到10倍，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：$F_e = 0.4$，$S_e = 10$，

根据Amdahl定律，加速比为：

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

$S_e = 100$？

$$S_n = \frac{1}{1 - 0.4 + \frac{0.4}{100}} = \frac{1}{0.604} = 1.66$$

例：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。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%。请比较这两种设计方案。

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

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

练习1：某计算机采用浮点运算部件后，使浮点运算速度提高到原来的20倍，而系统运行某一程序的整体性能提高到原来的5倍，试计算该程序中浮点运算所占的比例。

$$5 = \frac{1}{1-x+\frac{x}{20}} \quad \Rightarrow \quad x = 0.84$$

练习2：若将上题中部件的处理速度加快到100倍（即$S_e = 100$），则整个系统性能能够提高多少？

$$S_n = \frac{1}{1 - 0.84 + \frac{0.84}{100}} = \frac{1}{0.1684} = 5.94$$

3. CPU性能公式

- CPU的程序执行时间T_{CPU}
  - 程序执行的总指令条数IC（Instruction Counter）：取决于是指令集结构和编译技术
  - 平均每条指令的时钟周期数CPI（Cycles Per Instruction）：取决于计算机组成和指令集结构
  - 时钟主频 f_c：取决于硬件实现技术和计算机组成

$$T_{CPU} = IC \times CPI \times \frac{1}{f_c}$$

- n种指令，每种指令的时钟周期数CPI_i，出现次数I_i

$$T_{CPU} = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{f_c}$$

- 平均指令时钟周期数

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

例：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减为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)。

原系统的CPI=25%×4+75%×1.33=2

$$20\times2\%+x\times23\%+1.33\times75\%=2(CPI_{原})$$

方案1：使FPSQR操作的CPI为2

$$2\times2\%+x\times23\%+1.33\times75\%=?(CPI)$$

$$CPI=CPI_{原}-2\%\times(20-2)=2-2\%\times18=1.64$$

方案2：提高所有FP指令的处理速度

$$CPI=CPI_{原}-25\%\times(4-2)=2-25\%\times2=1.5$$

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=2/1.5= 1.33

4. 程序访问的局部性原理（Principle of Locality）

- 局部性分时间局部性和空间局部性
  - 时间局部性：程序中近期被访问的信息项很可能马上将被再次访问。
  - 空间局部性：指那些在访问地址上相邻近的信息项很可能会被一起访问。
  - 例如，程序执行时间的90%都是在执行程序中10%的代码；存储器体系的构成就是以访问的局部性原理为基础的

1.2.2 计算机设计者的任务

- 确定用户对计算机系统的功能、价格和性能要求

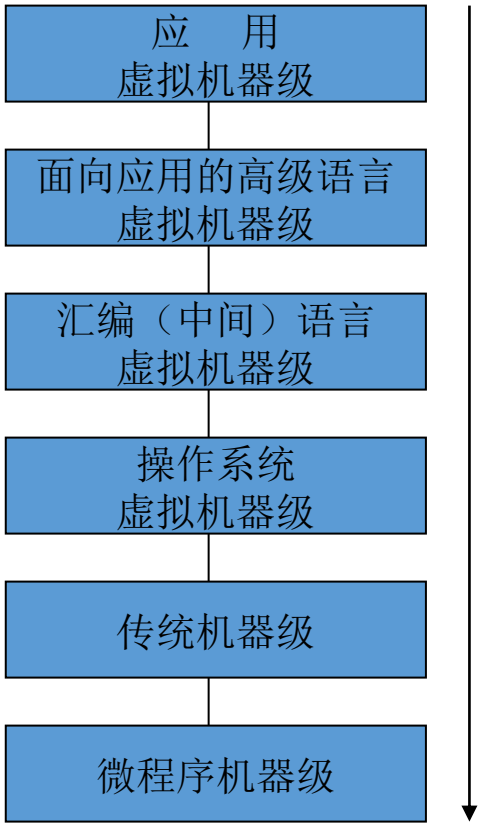
功能要求	应具备或支持的典型特性
应用领域 通用 科学计算 商用	决定对计算机系统的性能要求 对一系列任务有较好的性能 具有较好的浮点运算功能 支持COBOL、数据库、事务处理等功能
软件兼容级别 编程语言级 目标代码级	决定机器可以运行哪些软件 设计者的自由度较大，但需要新的编译器 系统结构已经确定，无须投资软件
操作系统要求 地址空间大小 内存管理 安全保护	为支持选定的操作系统所需要的特性 非常重要的特性，可能限制程序的运行 页式或段式等管理方式，现代操作系统需要 操作系统和应用程序需要
标准 浮点 I/O总线 编程语言 网络	市场上已有的，某种需要满足的标准 格式和算法：IEEE、DEC、IBM等 I/O设备：VME、SCSI、PCI、光纤等 影响指令集：C、FORTAN、COBOL等 对不同网络的支持：内部互连网、Ethernet等

- 软硬件平衡
  - 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
- 设计出符合今后发展方向的系统结构
  - 把握硬件和软件的发展趋势

1.2.3 计算机系统设计方法

方法1：由上向下（Top-Down）

- 设计过程：由上向下
  - 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
- 应用场合：专用计算机的设计
- 特点：对于所面向的应用领域，性能和性能价格比很高。随着通用计算机价格降低，目前已经很少采用



第一步：确定这一级的基本特性

第二步：设计或选择面向这种应用的高级语言

第三步：设计适于所用高级语言编译的中间语言

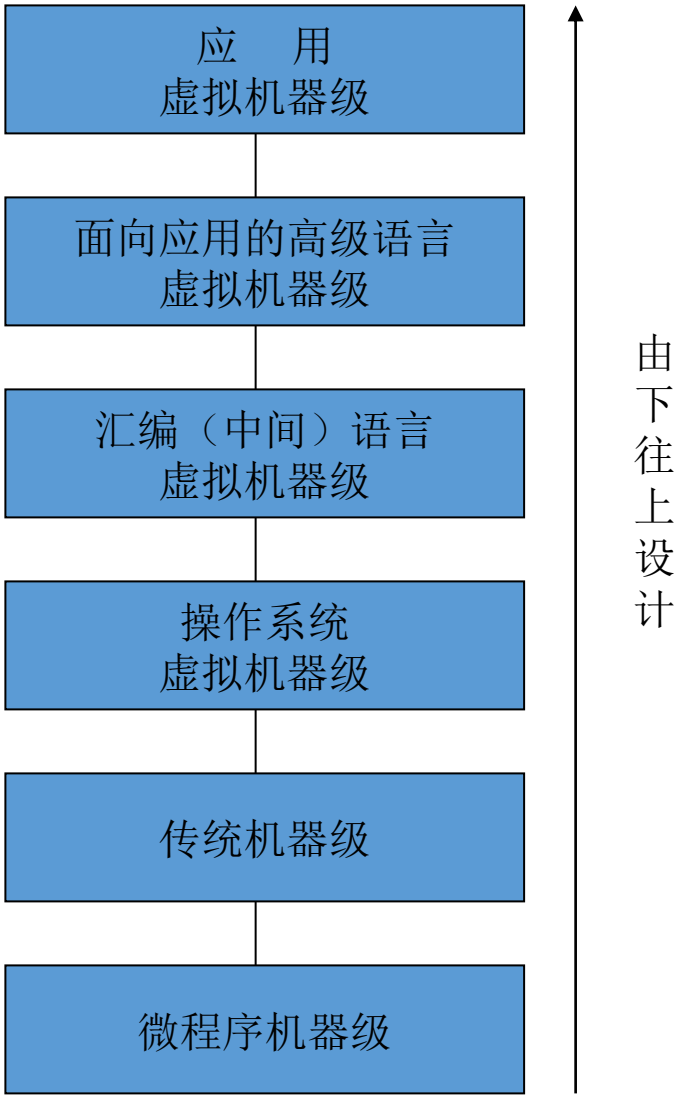
第四步：设计面向这种应用的操作系统

第五步：设计面向所用编译程序和操作系统的机器语言

第六步：设计面向机器语言的微指令机器硬件实现

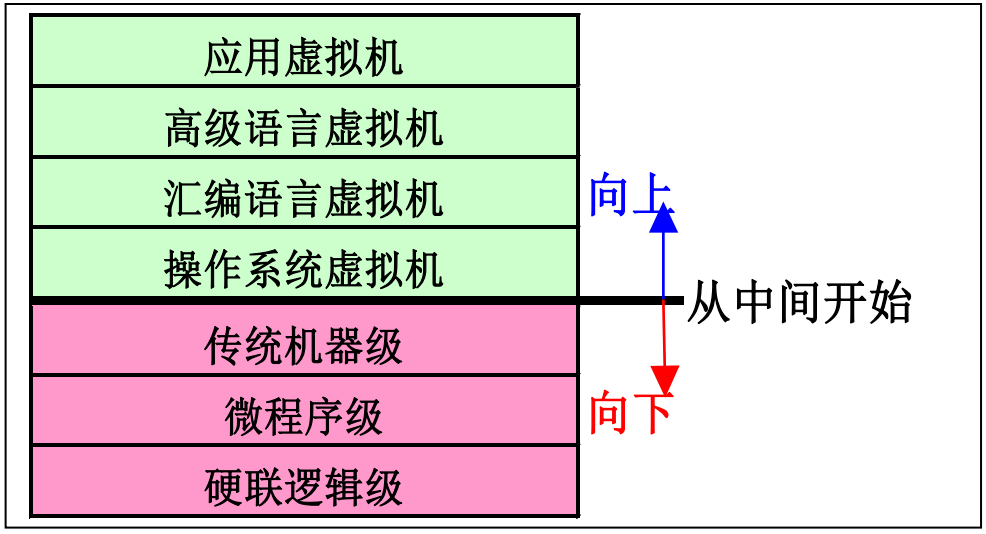
方法2：由下向上（Bottom-Up)

- **设计过程：**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多种操作系统、汇编语言、高级语言编译器等→最后设计面向应用的用户级
- **应用场合：**通用计算机的一种设计方法，在计算机早期设计中（60～70年代）广为采用
- **特点：**容易使软件和硬件脱节，整个计算机系统的效率降低。



方法3：中间开始（Middle-Out)

- **设计过程：**
  - 首先定义软硬件的分界面(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)
- **应用场合：**用于系列机的设计
- **特点：**软硬件人员结合、同时设计，软硬件功能分配合理。



1.3 计算机系统的评价标准

1.3.1 性能

- 主要标准：MIPS、MFLOPS、基准测试程序
- 性能比较

1.3.2 成本

1.3.1 性能

1. MIPS（Million Instructions Per Second）

$$\text{MIPS} = \frac{\text{指令条数}}{\text{执行时间} \times 10^6} = \frac{\text{Fz}}{\text{CPI}} = \text{IPC} \times \text{Fz}$$

- Fz为处理机的工作主频
- CPI（Cycles Per Instruction)为每条指令所需的平均时钟周期数
- IPC（Instruction Per Cycle)为每个时钟周期平均执行的指令条数

- 主要优点：直观、方便。目前还经常使用
- 主要缺点:

1） MIPS依赖于指令集，用MIPS来比较指令集不同的机器的性能好坏是很不准确的

2） 在同一台机器上，由于指令使用频度差别很大，MIPS会因程序不同而变化

3） MIPS可能与性能相反

例如，具有可选硬件浮点运算部件的机器，具有优化功能的编译器

2. MFLOPS（Million Floating Point Operations Per Second）

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

- 只能反映机器执行浮点操作的性能，并不能反映机器的整体性能（如编译性能）
- 基于浮点操作，比较适合用于衡量处理机中向量运算性能
- 会随着整数和浮点数的比例、快速浮点操作与慢速浮点操作的比例不同而不同
- 一般认为 $1\ \text{MFLOPS} \approx 13\text{MIPS}$

3. 用基准测试程序来测试评价机器的性能

- 采用实际的应用程序测试
  - 如：C语言的编译程序，CAD应用：Spice
- 采用核心程序测试
  - 从实际程序中抽出关键部分组合而成，例如循环部分
- 合成测试程序
  - 人为写的核心程序，规模小，结果预知
- 综合基准测试程序
  - 考虑了各种可能的操作和各种程序的比例，人为地平衡编制的基准测试程序

4 性能的比较

- 计算机的性能通常用**峰值性能**和**持续性能**来评价
- 峰值性能是指在理想情况下计算机系统可获得的最高理论性能值
- 持续性能又称为实际性能，它的值往往是峰值性能的5%~30%
- 持续性能的表示
  - 算数性能平均值
  - 调和性能平均值
  - 几何性能平均值

算数性能平均值

- 算数性能平均值A_m是n道程序运算速度或运算时间的算数平均值
- 以速度评价

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

- 以执行时间评价

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

- 加权算数平均（各个程序出现的比例不同）

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

调和性能平均值

- 调和性能平均值H_m（以时间评价）

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

- H_m的值与运行全部程序所需要的时间$\sum_{i=1}^{n} T_i$成反比，用他来衡量计算机的时间（速度）性能比较准确
- 加权调和平均

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

几何性能平均值

- 几何性能平均值G_m（以速度评价）

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

- 对不同机器进行性能比较时，可以对性能采取归一化处理，以某一台机器的性能作为参考标准，即R_i是第i个程序相对于参考机器归一化后的运行速率
- 几何平均速度与所参考的机器无关，不论哪台机器做参考机，G_m均能够正确反映结果的一致性：$G_m > 1$的机器性能相对较好，$G_m < 1$的机器的相对性能较差

$$\frac{G_m(X)}{G_m(Y)} = G_m\left(\frac{X}{Y}\right)$$

- 若以时间评价呢？

两个程序在三台机器上的执行时间

	机器 A	机器 B	机器 C
程序 P1（秒）	1	10	20
程序 P2（秒）	1000	100	20

运行程序P1时，A的速度是B的10倍；

运行程序P2时，B的速度是A的10倍；

运行程序P1时，A的速度是C的20倍；

运行程序P2时，C的速度是A的50倍；

运行程序P1时，B的速度是C的2倍；

运行程序P2时，C的速度是B的5倍。

两个程序在三台机器上的执行时间

	机器 A	机器 B	机器 C
程序 P1（秒）	1	10	20
程序 P2（秒）	1000	100	20

算术平均： $A_m(A) = 500.5, A_m(B) = 55, A_m(C) = 20$

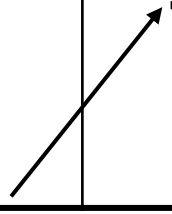
- 程序P1和P2各执行1次，B的速度是A的9.1倍；
- 程序P1和P2各执行1次，C的速度是A的25倍；
- 程序P1和P2各执行1次，C的速度是B的2.75倍。

结论：

- 执行程序P1和P2相同次数，机器A最慢，机器C最快
- 算术平均速度：三台机器的速度之比为：

A：B：C = 1：9.1：25

	机器A	机器B	机器C
程序P1执行时间(s)	1	10	20
程序P2执行时间(s)	1000	100	20
加权W1 (0.5, 0.5)	**500.50**	**55.00**	**20.00**
加权W2 (0.909, 0.091)	**91.91**	**18.19**	**20.00**
加权W3 (0.999, 0.001)	**2.00**	**10.09**	**20.00**



0.909×1+0.091×1000=91.91

加权算术平均

- 加权算术平均W1三台机器的速度：A < B < C
- 加权算术平均W2三台机器的速度：A < C < B
- 加权算术平均W3三台机器的速度：C < B < A

执行时间正交化	与A正交			与B正交			与C正交		
	A	B	C	A	B	C	A	B	C
程序P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
程序P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
算术平均	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
几何平均	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
总时间比	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

两个程序在三台机器上的执行时间

	机器 A	机器 B	机器 C
程序 P1（秒）	1	10	20
程序 P2（秒）	1000	100	20

几何平均

- 几何平均值与所参考的机器无关
  - 机器A与机器B的性能相同
  - 机器C的执行时间是机器A或机器B的0.63倍。
- 执行程序P1和P2的总时间，机器A几乎是机器B的10倍。

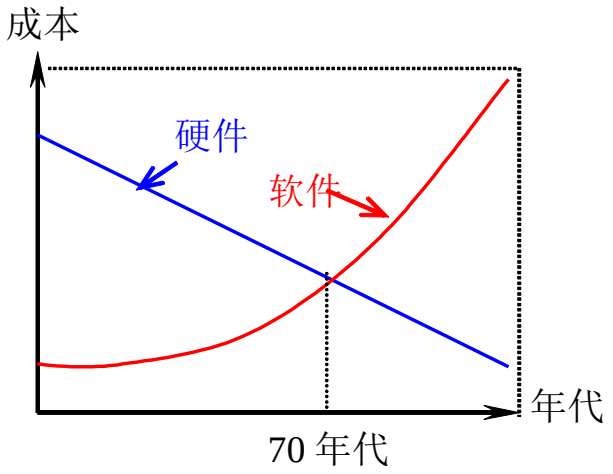
1.2.4 价格标准

➢价格与性能的关系：

- 速度每10年左右提高100倍，但价格基本维持不变
- 用当前同样的价格，在5年之后能买到性能高出10倍的计算机

➢硬件与软件的价格比例：

- 硬件在整个计算机系统价格中所占的比例在下降，软件所占的比例在上升
- 目前软件价格已经超过硬件价格



软件所占的成本越来越高

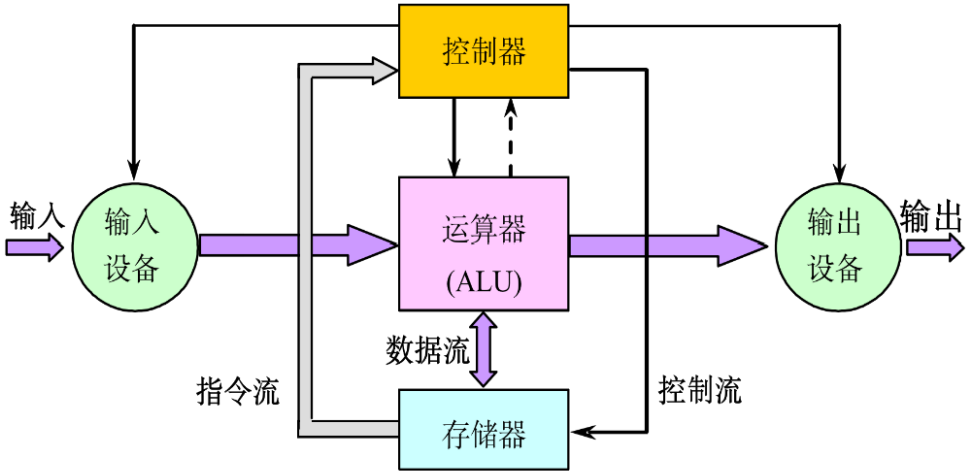
1.4 计算机系统的发展

1.4.1 冯·诺依曼结构
1.4.2 软件对系统结构的影响
1.4.3 价格对系统结构的影响 （课后阅读）
1.4.4 应用对系统结构的影响（课后阅读）
1.4.5 VLSI对系统结构的影响（课后阅读）
1.4.6 技术的发展对系统结构的影响（课后阅读）
1.4.7 算法和系统结构（课后阅读）

1.4.1 冯·诺依曼结构

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，计算机一旦启动，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



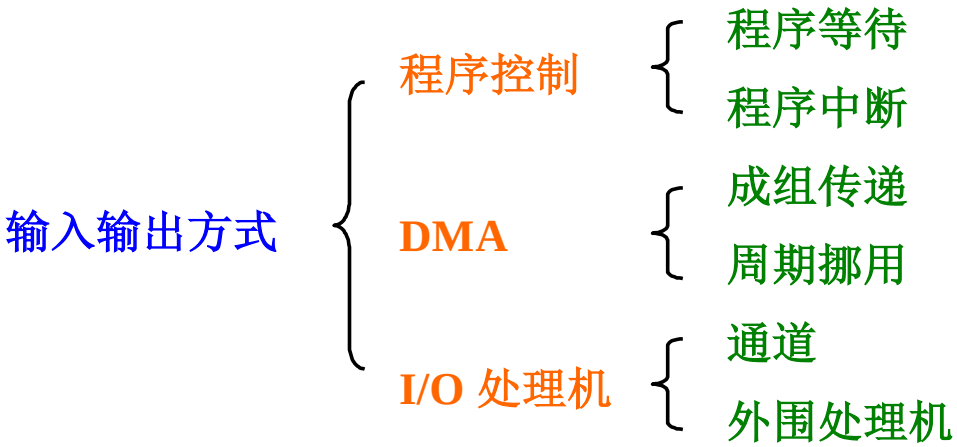
最早的存储程序式计算机是美国数学家冯·诺依曼（Von Neumann）等人与1946年总结并提出

2. 存储程序原理的基本点：指令驱动

- 以运算器为中心,集中控制。
- 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。
- 指令由操作码和操作数组成。
- 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
  - 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。

3. 对系统结构进行的改进

- 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
- 指令集的发展
  - 复杂指令集计算机（CISC）:20世纪70-80年代的计算机系统中，指令条数可达300-500条，寻址方式和指令格式的种类也很多
  - 精减指令集计算机（RISC）：1979年，D.A. Patterson等人提出了精减指令集计算机的思想，把指令集设计成只包含那些使用频率较高的少量指令，而且指令格式和寻址方式也很简单

1.4.2 软件对系统结构的影响

- 软件相对于硬件的成本越来越贵，已积累了大量成熟的系统软件和应用软件。人们希望在新的计算机系统结构出台后，软件能够继续在新的系统结构上运行，即要求软件具有可兼容性（或可移植性）
- 软件的可移植性（Portability）是指软件不修改或者只经过少量修改就可以由一台机器移到另一台机器上运行，同一软件可应用于不同的环境。

方法一：系列机方法

- 系列机定义:具有相同的系统结构，但组成和实现技术不同的一系列计算机系统
- 实现方法：在系统结构基本不变的基础上，根据不同的性能和不同的器件，研制出多种性能和价格不同的计算机系统。
- 一种系统结构可以有多种组成，一种组成也可以有多种物理实现，如IBM370系列机：115,125,135,145,158,168等

- 相同的指令系统，采用顺序执行、重叠、流水和其它并行处理方式
- 相同的32位字长，数据通路宽度为8位、16位、32位、64位。
- 如PC系列机有：
  - 不同主频：4.7MHz，500MHz，1GHz，2.4GHz, 3GHz, …
  - 不同扩展：Pentium、Pentium Pro、Pentium MMX、Pentium SSE、Pentium SSE2
  - 不同Cache：Pentium、Celeron、Xeon
  - 不同字长：8位、16位、32位、64位

- 采用系列机方法的主要优点：

  1） 系列机之间软件兼容，可移植性好

  2） 插件、接口等相互兼容

  3） 便于实现机间通信

  4） 便于维修、培训

  5） 有利于提高产量、降低成本

- 采用系列机方法的主要缺点：限制了计算机系统结构的发展，如PC系列机，其系统结构非常落后，使用也最普及

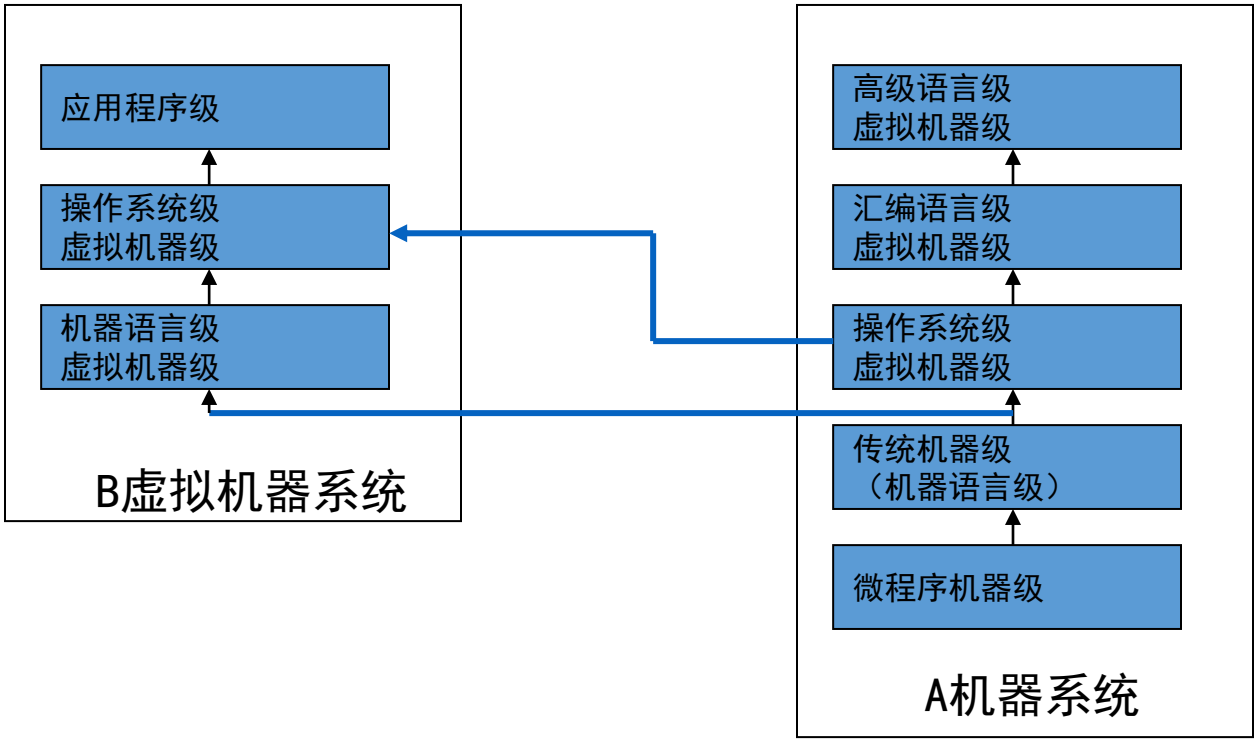
方法二：模拟和仿真

- 从指令系统来看，是指要在一种机器的系统结构上实现另一种机器的指令系统
- 模拟（Simulation）：用机器语言程序解释实现软件移植的方法
- 仿真（Emulation）：用微程序直接解释另一种机器指令的方法。

模拟 Simulation

- 用机器语言程序解释实现软件移植的方法。
  - 进行模拟工作的A机称为宿主机（Host Machine）
  - 被模拟的B机称为虚拟机（Virtual Machine）
  - 所有为各种模拟所编制的解释程序通称为模拟程序，编制非常复杂和费时
  - 只适合于移植运行时间短，使用次数少，而且在时间关系上没有约束和限制的软件；

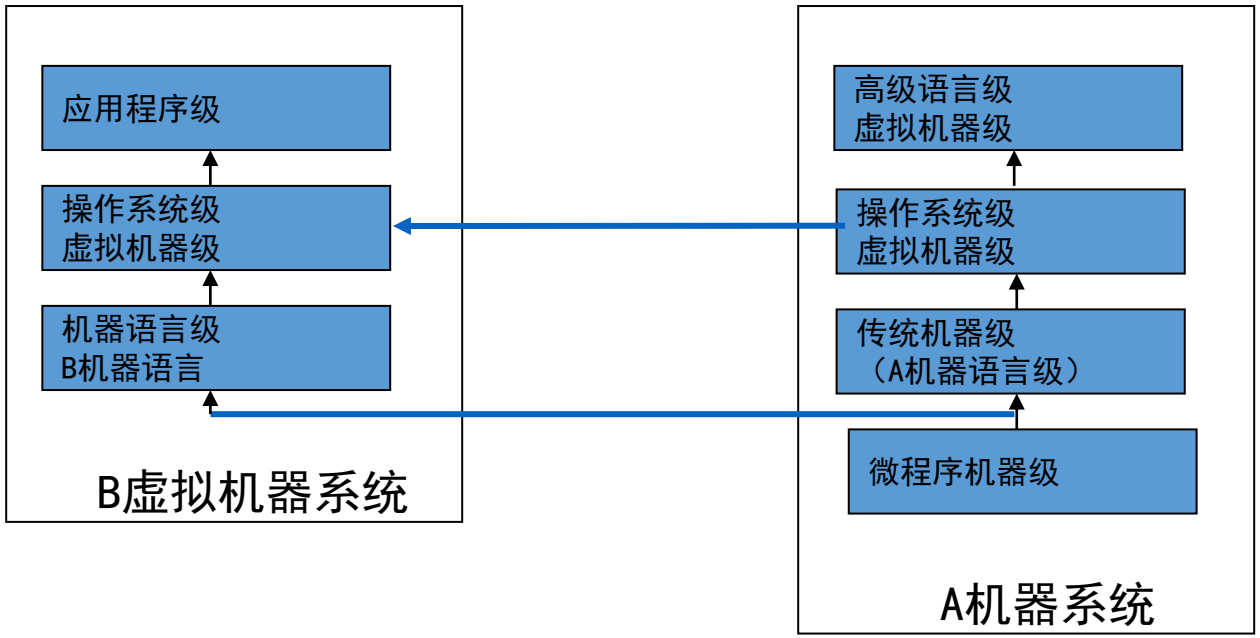
用模拟方法实现应用软件的移植



仿真 Emulation

- 用微程序直接解释另一种机器指令的方法。
  - 进行仿真工作的A机称为宿主机
  - 被仿真的B机称为目标机（Target Machine）
  - 所有为仿真所编制的解释微程序通称为仿真微程序；

用仿真方法实现应用软件的移植



仿真与模拟的区别

- 解释用的语言不同
- 解释程序所存的位置不同：仿真存在控制寄存器，模拟存在主存中
- 说明：
  - 模拟适用于运行时间不长、使用次数不多的程序
  - 仿真提高速度，但难以仿真存储系统、I/O系统，只能适用于系统结构差异不大的机器间；
  - 在开发系统中，两种方法共用

方法三：统一高级语言

- 实现方法：采用同一种不依赖于任何具体机器的高级语言编写系统软件和应用软件
- 困难：至今还没有这样一种高级语言，短期内很难实现。C、Ada、Java、……

三种方法比较：

- 采用统一高级语言最好，是努力的目标
- 系列机是暂时性方法，也是目前最好的方法
- 仿真的速度低，芯片设计的负担重，目前用于同一系列机内的兼容，1/10～1/2的芯片面积用于仿真

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

- **并行性（Parallelism）**：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
  - 同时性（Simultaneity）：两个或两个以上的事件在同一时刻发生
  - 并发性（Concurrency）：两个或两个以上的事件在同一时间间隔内发生

1.5.1 并行性的概念

- 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或程序

1.5.2 提高并行性的技术途径

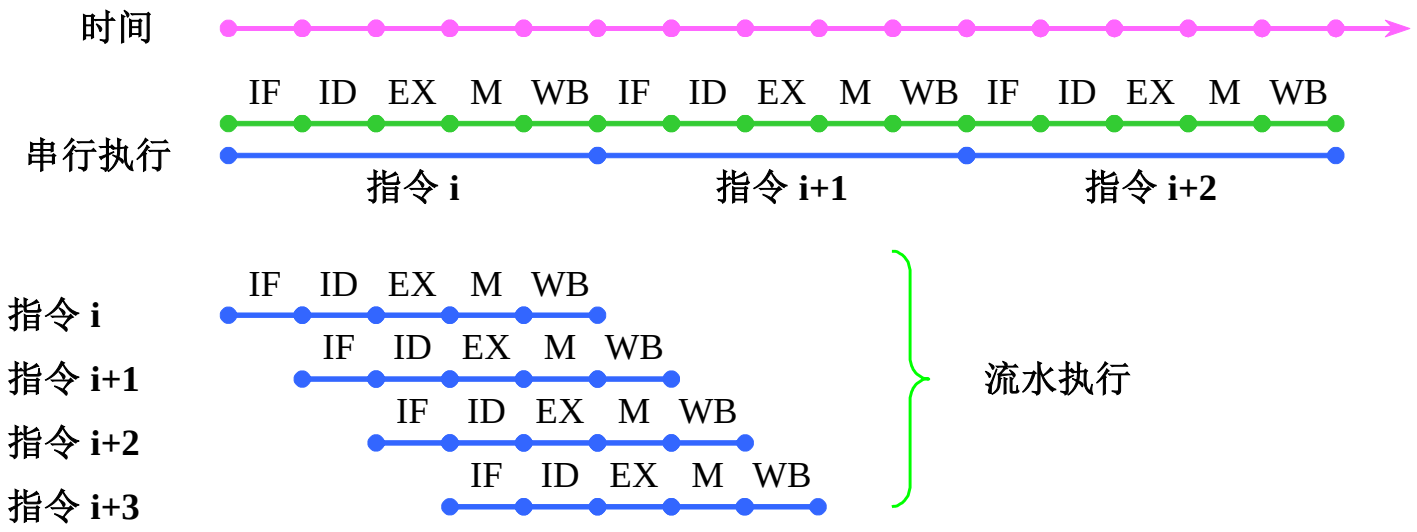
- **时间重叠**：引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度
- **资源重复**：引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能
- **资源共享**：这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。

1.5.3 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；
  - 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

1.5.3 单机系统中并行性的发展

例如：对于解释指令的5个过程，就分别需要5个专用的部件：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)和写结果部件(WB)



1.5.3 单机系统中并行性的发展

- 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
  - 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 。

1.5.3 单机系统中并行性的发展

- 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
  - 阵列处理机（并行处理机）
  - 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。

1.5.3 多机系统中并行性的发展

- 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：**异构型多处理机**、**同构型多处理机**、**分布式系统**
- 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的

1.5.3 多机系统中并行性的发展

- 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。

1.5.3 多机系统中并行性的发展

- 机间互连 （资源重复）
  - 容错系统
  - 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。

本章重点：

1. 计算机系统的层次结构

2. 计算机系统结构的定义及研究内容

3. 计算机系统的评价方法

4. 冯·诺依曼结构及其发展

5. 透明性、系列机、兼容性等概念

6. 计算机系统结构并行性

7. 了解计算机系统的分类方法