第一章 概论

1.1 多级层次概念和机器级实现技术

- **计算机系统**：由一台或多台计算机和相关软件组成并完成 某种功能的复杂系统
- 多级层次结构
  - **概念**：从使用语言的角度，将计算机系统按功 能划分为多级层次结构，以对整个系统 进行描述、分析、设计和使用。
  - **分类**
    - 数字逻辑 —— 硬联逻辑
    - 微程序级 —— 微程序机器
    - 机器语言级 —— 传统机器
    - 操作系统级 —— 操作系统机器
    - 汇编语言级 —— 汇编语言机器
    - 高级语言级 —— 高级语言机器
    - 应用语言级 —— 应用语言机器
  - **优点**
    - 有利于理解软件、硬件、固件的地位和作用
    - 理解各种语言的实质和实现途径
    - 探索虚拟机新的实现方法和新的系统设计 □
      - 例如，由硬件和固件实现高级语言机器，由真 正的微处理机代替虚拟机，实现多处理机、分布处理、计算机网络等体系结构
    - 在一台真正的宿主机上通过模拟或仿真另一 台不同的假想机器——虚拟化技术
    - 理解计算机体系结构的定义
- **机器级的实现技术**：实际上，只有二进制机器指令（即机器 语言）可以直接被硬件识别和执行，其 上面各层机器级所使用的语言都必须被 翻译或解释为较低层机器级上语言，由 较低层机器实现
- **翻译**：先将高一级机器级上 的程序整个地变换成 低一级机器级上可运 行的等效程序，然后 再在低一级机器级上 去实现的技术。在执 行过程中，高一级程 序不再被访问。
- **解释**：每当一条高一级指令 被译码后，就直接去 执行一串等效的低一 级指令，然后再去取 下一条高一级的指令， 依此重复进行。
- 解释比翻译费时，但节 省存储空间
- 翻译 - 翻译 - （翻译+解释）- 解释 - 硬件

1.2 计算机体系结构

- **定义**：基于多级层次结构：计算机体系结构是对各机器级界面的划 分、定义及上下级功能分配。 按照计算机系统的多级层次结构，不同 机器级的界面有很大不同，意味着每个 机器级都有其系统结构。
- **实质**：计算机体系结构概念的实质是计算机系统中软硬件界面（指令集结构 ISA）的确定，其界面之上的是软件的功能， 界面之下的是硬件和固件的功能。

- **透明性**：本来存在的事物或属性，从某个角度看 却好象不存在。
- **设计内容**
  - 数据通路宽度： 数据总线上一次并行传输的信息的位数。
  - 专用部件设置： 是否设置乘法、浮点等专用部件，其数量与性能、 价格、专用部件的使用频度有关
  - 操作对部件的共享程度：共享程度高：价格便宜，但减低速度；共享程度低：并行度高， 提高速度，价格高
  - 功能部件的并行度：顺序串行，或 重叠、流水，或 分布处理；
  - 控制机构的组成方式：硬联，或 微程序控制，或 单机/多机/功能分布处理；
  - 缓冲和排队技术： 缓冲器及容量； 处理顺序：随机、先进先出、 先进后出、优先级、循环等
  - 预估、预判技术： 预测未来行为的原则，以优 化性能。
  - 可靠性技术等：冗余、容错等，以提高可靠 性。
- **目标**：计算机组成的设计目前主要围绕提高速度，着重 提高并行度、重叠度，以及分散 功能和设置专用部件来进行。

1.3 软硬件取舍与计算机系统设计思路

- **软 or 硬：**
  - 在逻辑功能上，软件和硬件是等效的。
  - 原理上，软件实现的功能完全可以由硬 件实现，硬件实现的功能也完全可以由 软件模拟完成
  - 系统结构的一个主要任务就是进行软硬 件功能分配
- **三大原则**
  - 在现有器件和技术条件下，系统要有高 的性能价格比。
  - 要考虑到准备采用和可能采用的组成技 术，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
  - 为软件(如OS、编译、高级语言、应用 软件等)的设计和实现提供更多更好的硬 件支持。
- **三种设计思路**
  - **由上往下**：从应用开始，确定好面向 使用者的那级虚拟机器的基本特 性和环境，然后逐级往下设计。
    - 优点
      - 运行效率高，软硬分配合理
      - 适用于专用机的设计
    - 缺点
      - 适应性差，周期长，软、硬件脱节
    - 解决
      - 不完全优化
      - 不专门设计，从已有机器中“选型”
  - **由下往上**：不考虑应用，根据器 件等情况研制硬件，根据要 求配置软件。
    - 优点
      - 可设计通用计算机
    - 缺点
      - 硬件无法改变，软件设 计被动；
      - 软、硬件脱节；
      - 某些性能指标是虚假的

- **从中间开始**：从软硬件界面开始，同 时进行软硬件设计。目前多将 “中间”取在传统机器级与操 作系统机器级的界面上
  - 优点
    - 软硬件功能分配比较合理
    - 缩短了研制周期
    - 交互式设计方法
  - 缺点
    - 对设计人员要求高
    - 应有有效的软件设计环境和开发工具

1.4 计算机设计的量化准则

- 性能指标
  - 响应时间：包括CPU时间（运行一个程序所花费的 时间）与等待时间（用于磁盘访问、存 储器访问、I/O操作、操作系统开销等 时间）。
    - CPU时间=用户CPU时间+系统CPU时间。
      - 用户CPU时间：用户程序所花费的CPU时间。
      - 系统CPU时间：用户程序运行期间操作系统 花费的CPU时间(例如在内核内执行内存copy 系统服务所花费的时间)。
  - 吞吐率（MIPS）：$\frac{指令条数}{CPU_{时间}*10^6}$
  - MFLOPS: $\frac{程序中的浮点操作次数}{CPU_{时间}*10^6}$
  - 1 mflops = 3 mips(约等于)
  - CPU性能公式：
    - 主时钟频率f_c
    - 程序执行过程中所处理的总指令数IC
    - 平均每条指令时钟周期数CPI
    - 每个时钟周期执行的指令数$IPC=\frac{1}{CPI}$
    - $T_{CPU}=\frac{IC*CPI}{f_c}=IC*CPI*T_{CLK}=\frac{CLK}{f_c}=CLK*T_{CLK}$(cpu全速工作时完成该进程的时间)
    - 等效指令速度：
      - 等效指令执行时间 $T=\sum_{i=1}^{n}(W_i\times T_i)$
      - 等效指令执行速度 $MIPS=\frac{1}{\sum_{i=1}^{n}\frac{W_i}{MIPS_i}}$
      - 等效$CPI=\sum_{i=1}^{n}(CPI_i\times W_i)$
      - W_i为第i种指令的使用频度 Ti为执行时间
- Amdahl定律

- 系统中对某一部件采用某种更快执行方式所能获得的系统性能改进程度，取决于这种执行方式被使用的频率或所占总执行时间的比例。
- 加速比$S_n = \frac{改进后性能}{改进前性能}$
- 可改进比例$Fe = \frac{T_1}{T_{old}}$ 0~1
- 部件加速比$Se = \frac{T_1}{T_2}$ Se > 1
- 性能递减规则：如果仅仅对计算机中的一部分做性能改 进，则改进越多，系统获得的效果越小
  - 推论：：如果只针对整个任务的一部分进 行优化，那么所获得的加速比不大于 1/(1 -Fe) (即Se->∞时)。
- 结论：一个"好"的计算机系统： 是一个各部分性能均能平衡得到提高的 系统，而不能只是其中一个功能部件性 能的提高。

1.5 影响系统结构的因素

- 软件对系统结构的影响——解决软件的可移植性问题
  - **可移植性定义**：软件不用修改或只需少量加工就能由一 台机器搬到另一台机器上运行，即同一 软件可以应用于不同的环境。
  - 实现技术
    - 统一高级语言：采用统一的、不依赖于任何具体机器的 通用高级语言编制应用程序，则应用软件可以移植于不同的机器上
    - 采用系列机思想：统一汇编语言或机器语言，实现系统结 构相同或相似的机器之间的软件移植（与"从中间开始"的设计方法呼应）
    - 模拟与仿真：
      - 模拟：用一种机器(A)的机器语言解释实现另一种 机器(B)的指令系统，使A具有B的指令系统，从而实现软件移植的方法（灵活性大）
        - 缺点
          - 模拟程序很复杂、编制费时；
          - 不能被硬件直接执行，速度较低、实时性差；
          - 适用于运行时间短、使用次数少，而且没有 时间约束和限制的软件。
      - 仿真：用一种机器(A)的微程序直接解释实现另一 种机器(B)的指令系统，从而实现软件移植 的方法
      - 比较
        - 仿真：使用微程序，解释程序在控存中
        - 模拟：使用机器语言，解释程序在主存中
        - 模拟方法运行速度低，仿真方法速度高
        - 仿真需要较多的硬件（包括控制存储器）
        - 系统结构差别大的机器难于完全用仿真方法 来实现
        - 通常将模拟和仿真混合使

1.6 体系结构中的并行性

- **定义**：可以同时进行是提高系统性能的有效途径，包含两重含义：

    - 同时性(simultaneity)：同一时刻发生
    - 并发性(concurrency)：同一时间间隔内 发生
- **分类**:
    - 从处理数据的角度:
        - 位串字串：一个字的一位，串行单处理机
        - 位并字串：一个字的全部位，并行单处理机
        - 位串字并：许多字的同一位，并行处理领域
        - 全并行：许多字的全部或部分位组
    - 从信息加工的步骤和阶段
        - 存储器操作并行：在一个存储周期内访问多个字 (单体多字)，进而实现相联访问(多体多字) -> 并行存储器系统，相联存储器系统，相联处理机；
        - 处理器操作步骤并行：将操作步骤或具体操作的执行步骤在时间上重叠流水地进行 -> 流水线处理机；
        - 处理器操作并行：重复设置大量处理单元，在同 一个控制器的控制下，对多个数据组同时操作 -> 阵列处理机；
        - 指令、任务、作业并行：多个处理机同时对多条指令及有关的多数据组进行处理 -> 多处理机
    - 实现方法
        - 时间重叠：引入时间概念，让多个处理过程轮流使 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，重复设置多套硬件 。提高速度，同时提高可靠性
        - 资源共享：利用软件让多个用户轮流使用同一套资源
            - 例如：多道程序分时系统，多处理机， 计算机网络，分布处理系统等；

1.7 计算机系统的分类方法

- 传统分类方法
    - 按大小划分
        - 种类：巨型机、大型机、中型机、小型机、 微型机等
        - 划分原则：以性能为表征，按价格来划分
        - 存在问题：划分的标准是随时间而变化
    - 按用途划分
        - 种类：科学计算、事务处理、实时控制、工 作站、服务器、家用计算机等。
        - 划分原则：
            - 科学计算：浮点计算速度；
            - 事务处理：字符处理、十进制运算；
            - 实时控制： 中断响应速度、I/O能力；
            - 工作站：图形处理能力。
        - 发展方向：具备上述所有功能的通用处理机
    - 按数据类型划分
        - 种类：定点机、浮点机、向量机、堆栈机等
    - 按处理机个数和种类划分

        - 种类：单处理机、并行处理机、多处理机、 分布处理机、关联处理机、超标量处理机、 超流水线处理机、SMP（对称多处理机）、 MPP（大规模并行处理机）、机群 （Cluster）系统等

- 弗林分类法
    - 概念
        - 指令流：机器执行的指令序列
        - 数据流：指令流调用的数据序列
        - **多倍性**：在系统瓶颈部件上处于同一执行阶段的指令或数据的最大可能个数
    - 分类
        - **SISD系统(单指令流，单数据流系统)**：每次只对一条指令译码，指令部件一次仅处理一 条指令，仅对一个操作部件分配数据；
            - 典型：传统的单处理机，流水方式的单处理机。
        - **SIMD系统(单指令流，多数据流系统)**：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；
            - 典型：阵列处理机，相联处理机，流水线处 理机。
        - **MISD系统(多指令流，单数据流系统)**：n个操作部件按n条不同指令的要求，对同 一个数据流及中间结果分别进行不同的处理；
            - 典型：少见。宏流水，脉动阵列处理机。
        - **MIMD系统(多指令流，多数据流系统)**：实现作业、任务、指令、数组等各级全面并 行的多机系统；
            - 典型：多处理机，多计算机系统；
    - 评价
        - 优点
            - 能反映出大多数计算机的并行性 工作方式和结构特点。广泛使用。
        - 缺点
            - 只针对控制流型机器分类，对流 水线处理机的分类不明确，难以反映出 计算机系统在工作原理上的差别。
            - 分类太粗：例如，在SIMD中包括有多种处 理机，对流水线处理机的划分不明确，标量 流水线为SISD，向量流水线为SIMD。
            - 把两个不同等级的功能并列对待；通常，数 据流受指令流控制，从而造成MISD不存在

第二章 数据表示与指令系统

2.1 数据表示与数据结构

- 数据表示的定义：可以被硬件直接识别和指令系统直接调用的 数据类型。
- 数据结构的定义：结构化数据的组织方式，反映了应用中各种 数据元素或信息单元之间的结构关系。
- 二者的关系
    - 数据结构是通过软件映像成机器所具有的各 种数据表示实现的；
    - 数据表示是数据结构的组成元素；
    - 数据表示为数据结构提供不同程度的支持， 反映在效率和方便程度的不同；
    - 数据结构和数据表示都是数据类型的子集。

- 哪些数据类型用数据表示实现，哪些用数据结 构实现，实质上是一个软、硬件取舍的问题。
- 目前，定点数、浮点数、逻辑数、十进制数、 字符串等基本数据表示和变址操作是不可缺少 的。但还需要为数据结构提供更进一步的支持， 减少软件负担。

- 高级数据表示
  - **自定义数据表示**
    - 定义：由数据本身来表明数据类型，使计算机 内的数据具有自定义能力。
    - 目的：缩短机器语言与高级语言对数据 属性的说明之间的语义差距。
    - 分类
      - **带标志符的数据表示**：让每个数据都带有 类型标志位
        - 标志符由编译器或其它系统软件建立， 对程序员透明。
        - 将数据类型与数据本身直接联系在一起， 使机器语言中的操作码与高级语言中的 运算符一致。
        - 优点
          - 提高了操作码的通用性，简化了指令系统和程序 设计；
          - 简化的编译程序；
          - 便于实现一致性检查，可由硬件直接快速地检测 出多种程序错误；
          - 能够由硬件自动完成数据类型转换；
          - 支持了数据库系统的实现与数据类型无关的要求；
          - 为软件调试和开发提供了支持。
        - 缺点
          - 可能会使程序占用的空间增加。需要合理 设计和使用；
          - 单条指令的执行速度可能会降低。但编制、 调试的时间缩短，总时间缩短。
      - **数据描述符**
        - 定义：描述数据块的属性，与数据分开存放。
        - 目的：支持向量、数组、记录等数据， 减少标志符所占用的空间。
        - 与带标志符的数据表示的区别
          - 标志符要与每个数据相连，两者合存在一个 存储器单元中；而描述符则和数据分开放；
          - 要访问数据集中的元素时，必须先访问描述 符，这就至少要增加一级寻址；
          - **描述符可看成是程序一部分，而不是数据一 部分**，因为它是专门来描述要访问的数据的 特性。
        - 优点
          - 将描述符按树形连接，可以描述复杂和多维 数据结构；
          - 为向量、数组等数据结构的实现提供了一定 的支持，有利于简化编译，可以比变址法更 快地形成元素地址。
  - 向量数组数据表示
    - 向量
      - 定义：指具有n个数据的数组。
      - 特点
        - 各个数据称为数组的元素；

            - 每个数据具有相同的数据类型，(如实数或 逻辑数)；相同的数据表示（如字长、字的 格式相同）；进行相同的操作；
            - 各数据之间是独立无关的。
        - 向量处理机：，在硬件上设置有丰富 的向量或阵列运算指令，配置有以流水 或阵列方式处理的高速运算器。
        - 优点
            - 快速形成元素地址；
            - 实现数据块的预取；
            - 用一条向量、数组指令，借助流水和并行运 算等处理方式，可以同时实现对整个向量、数组的高速处理；
            - 硬件处理稀疏向量和交叉阵列，节省存储空 间，节省处理时间
    - 堆栈数据表示
        - 有由若干高速寄存器组成的硬件堆栈，通过附加 控制电路，让它与主存中的堆栈区在逻辑上组成一个整体，使堆栈的访问速度是寄存器的，但容 量是主存的；
        - 有丰富的堆栈操作类指令，功能很强，可以直接 对堆栈中的数据进行处理；
        - 有力地支持高级语言的编译，简化了编译，缩小 了高级语言与机器语言的语义差距；

2.2 寻址方式

- 编址方式
    - 定义：指对各种存储设备进行编码的方法
    - 常用编址单位：字 / 字节 / 位 / 块
    - 编址方式
        - 分类编址：将部件适当分类，每类各自从“0”开始单 独编址。（通用寄存器 & 主存储设备 & 输入输出设备）
        - 统一编址：把各种部件统一编成一个从“0”开始的一 维线性地址空间（一个零地址空间）。
        - 隐含编址：采用事先约定的编址方式隐含寻址，无地址空间，访问比较快，但可能使指令设计不规范
- 寻址方式
    - 定义：指令寻找（或访问）到所需要的数据的方法
    - 方式（抽象）
        - 面向寄存器
            - 操作数可以取自寄存器或主存。
            - 结果大多数送寄存器，少数送主存。
        - 面向主存
            - 主要访问主存，少量访问寄存器。
        - 面向堆栈
            - 主要访问堆栈，少量访问主存或寄存器。
            - 支持高级语言，有利于编译程序； 节省存储空间；支持程序的嵌套和递归调用， 支持中断处理。
    - 方式（具体）

寻址方式	指令实例	含　　义
寄存器寻址	Add R4 , R3	Regs[R4]←Regs[R4] + Regs[R3]
立即数寻址	Add R4 , #3	Regs[R4]←Regs[R4] + 3
偏移寻址	Add R4 , 100(R1)	Regs[R4]←Regs[R4] + Mem[100+Regs[R1]]
寄存器间接寻址	Add R4 , (R1)	Regs[R4]←Regs[R4] + Mem[Regs[R1]]
直接寻址或绝对寻址	Add R1 , (1001)	Regs[R1]←Regs[R1] + Mem[1001]
存储器间接寻址	Add R1 , @(R3)	Regs[R1]←Regs[R1] + Mem[Mem[Regs[R3]]]
自增寻址	Add R1 , (R2)+	Regs[R1]←Regs[R1] + Mem[Regs[R2]] Regs[R2]←Regs[R2] + d
自减寻址	Add R1, -(R2)	Regs[R2]←Regs[R2] − d Regs[R1]←Regs[R1]+Mem[Regs[R2]]
缩放寻址	Add R1 , 100(R2)[R3]	Regs[R1]←Regs[R1] + Mem[100 + Regs[R2] + Regs[R3]*d]

- 程序在主存中的定位技术
  - 直接
  - 静态再定位：用软件的方法把目的 程序的逻辑地址变换成物理地址。
  - 动态再定位：在执行指令时，才形成访问主存的物理 地址；
    - 程序执行时，通过例如地址加法器将逻 辑地址加上基址寄存器的内容(程序基点 地址)，形成物理地址，然后访存，地址 变换速度快。
- 信息的存储方式
  - 任意存储
  - 按字节数或位数的整数倍存储

2.3 指令系统的设计和优化

- 指令操作码的优化
  - 目的：在足够表达全部指令的前提下，使操作码字段所占用的位数少，从而缩短指令字长，减少程序所占存储空间。
  - 编码方法
    - 一些概念：
      - 信息源熵：$H = -\sum p_i log_2 p_i$，其中p位第i个操作吗出现的概率
      - 信息冗余量：$R = 1 - \frac{H}{\text{实际平均码长}}$
        - 实际平均码长$L = \sum p_i l_i$，其中l为第i个操作码的长度
    - 固定长度编码
      - 定长：所有指令的操作码长度相同。
      - 优点：规整，便于译码。
      - 缺点：信息冗余量大，指令长，程序大。
    - Huffman编码

            - **基本思想**：当各种事件发生的概率不均等时， 使用概率高的事件用短代码表示，使用概率 低的事件用长代码表示，就会使平均位数缩 短。
            - 前提条件：**必须事先知道事件发生的概率。**
            - 特点：哈夫曼编码并不唯一，但平均码长唯一。
            - 优点：实际平均码长短。
            - 主要缺点：
                - 操作码长度很不规整，硬件译码困难。
                - 与地址码共同组成固定长的指令比较困难。
        - 扩展编码
            - 操作码长度不固定，但只有有限的几种长度。
            - 使用频度高的指令用短操作码表示，使用频度低的指令用长操作码表示(哈夫曼压缩思想)。
            - 一般采用等长扩展，以便于译码。
            - 等长扩展编码，每次扩展的位数相等， 便于实现分级译码。
            - 有多种不同的等长扩展方法，如：
                - 4-8-12 （位数）
                - 15/15/15 （条数）
                - 8/64/512（条数）等
            - 无论是Huffman还是扩展操作码的编码方法， **都不允许短码是长码的前缀**
- 指令字格式优化
    - 目的
        - 用短的位数来表示指令的操作信息和地址 信息，使程序中指令的平均字长短；
        - 增加指令字所能表示的操作信息和地址信息， 如寄存器数目和寻址方式类型；
        - 能够在具体实现中容易处理。
    - 基本思想：可变长操作码与可变长地址 码配合。
    - 方法：通常采用长操作码配短地址码。
        - 指令字长度固定
            - 采用多种地址制：由于各种指令所需要 的操作数的个数会有不同，因此指令系 统可以采用多种地址制，例如：
                - 三地址制
                - 二地址制
                - 一地址制
                - 零地址
        - 指令字长度可变：采用具有多种指令字长度的指令系统。
            - 能有效地减少指令集的平均指令长度， 降低目标代码的长度。
            - 使得各条指令的字长和执行时间大不一 样。

2.4 指令系统的发展与改进

- 方向
    - 增强指令系统的功能

- 简化指令系统

- CISC 复杂指令集系统
  - 功能
    - 进一步增强原有指令的功能
    - 设置更为复杂、但功能更强的新指令以取代原先 由软件子程序完成的功能，实现软件功能的硬化。
    - 按这种途径和方向发展，会使机器的指令系统越 来越庞大和复杂，因此称采用这种途径设计而成的CPU的计算机为复杂指令集计算机(CISC - Complex Instruction Set Computer)。
    - 目前多数计算机属于CISC型，如IBM 370， Intel i486，MC68040等。
  - 目标：强化指令功能，减少程序的指令条数，以达 到提高性能的目的。
  - 实现
    - 面向目标程序的优化实现
      - 目的
        - 减少目标程序占用的存贮空间；
        - 提高程序的运行速度(减少访存次数，缩短 指令执行时间)；
        - 更容易实现
      - 方法
        - 利用哈夫曼思想改进指令：频度高就增强功能、加快执行速度、缩短指令长度、增设新指令来替代；频度低就想办法合并或者取消
        - 增设**强功能**复合指令：存、取、条件转移（如增设组合转移）
      - 特点
        - 不删改原有的指令系统；
        - 增加少量强功能新指令替代常用指令串或子 程序； □满足软件向后兼容的要求
    - 面向高级语言的优化实现
      - 目的
        - 缩短高级语言与机器语言的语义差异，以利于支持高级语言编译系统；
        - 缩短编译程序的长度和编译所需要的时间。
      - 方法
        - 主要改进方法：
          - 统计使用频度来改进指令；
          - 面向编译、优化代码生成来改进指令；
          - 改进指令系统，缩小与各种语言的语义差异；
          - 让机器具有多种指令系统，多种系统结构；
          - 发展高级语言计算机；
    - 面向操作系统的优化实现
      - 目标
        - 缩短OS与系统结构语义差距；
        - 减少运行OS所需要的辅助操作时间；
        - 节省OS占用的存储空间。
      - 方法
        - 统计使用频度来改进；

        - 增设专用于OS的新指令；
        - 用硬件或固件实现OS的某些功能；
            - 把OS中使用频繁、对速度影响大的子程序进 行硬化或固化，直接由硬件或微程序实现。
        - 由专门的处理机完成OS，实现功能分布处 理系统结构。
    - CISC存在的问题
        - CISC中，大约20%的指令占据了80%的处理机时间。
        - 软硬件的功能分配问题——复杂的指令使指令的执行周期大大加长
- RISC 精简指令集
    - 目的：通过精简指令，使计算机结构变得简单、 合理、有效，并克服CISC结构的缺点。 CISC增强了指令系统功能，简化了软件， 但硬件复杂了。
    - 原则 —— **减少CPI是精华**
        - 只选择使用频度很高的指令，增加少量 支持操作系统和高级语言等的有用的 指令。指令条数一般<100条；
        - 减少寻址方式的种类，一般<2 种；
        - 简化指令格式，一般限制在2种以内，并 让所有指令有相同的长度；
        - 所有指令都在一个机器周期内完成；
        - 扩大通用寄存器的个数，一般≥32个， 尽可能减少访存，除STORE和LOAD指 令外，其他指令的操作都在寄存器之间 进行；
        - 为提高速度，大部分指令都采用硬联控 制实现，少量可采用微程序实现；
        - 通过精简指令和优化设计编译程序，以 简单有效的方式支持高级语言的实现。
    - 优点
        - 简化了指令系统设计，适合于VLI实现；
        - 提高了执行速度和效率；
        - 减低了成本，提高了可靠性；
        - 可以提供直接支持高级语言的能力，简化了 编译程序的设计；
    - 缺点
        - 指令少，指令功能简单，大了程序占用空 间，加重了汇编程序员的负担，加大了指令 信息流量；
        - 对浮点运算和虚拟存储器的支持很强大，但 不够理想；
        - 比CISC的编译程序难写。
    - 关键技术
        - 重叠寄存器窗口技术
        - 流水线技术与延时转移技术
        - 指令取消技术
        - 指令流调整技术
        - 优化编译系统设计的技术
- RISC和CISC的对比

	CISC	RISC
价格	硬件复杂，芯片成本高	硬件较简单，芯片成本低
性能	减少代码尺寸，增加指令的执行周期数	使用流水线降低指令的执行周期数，增加代码尺寸
指令集	大量的混杂型指令集，有专用指令完成特殊功能	简单的单周期指令，不常用的功能由组合指令完成
应用范围	通用机	专用机
功耗与面积	含有丰富的电路单元，功能强、面积大、功耗大	处理器结构简单，面积小，功耗小
设计周期	长	短

第三章 输入输出系统

3.1 概述

- 定义：在计算机系统中，把处理机与主存储器之外的 部分统称为输入输出系统
- 功能
  - 对指定外设进行I/O操作，同时完成许多其它 的管理和控制任务，如：
    - 编址
    - 准备通路
    - 信息传送
    - 格式变换
    - 中断请求
- 内容：输入输出设备、设备控制器、与输入输出操作有关的软硬件
- 设计目标：成本/性能/支持多种设备，同时避 免I/O性能瓶颈。
- 操作
  - 在低档单用户计算机上
    - 由程序员自己安排。
    - I/O系统设计主要考虑解决好CPU、主存和 I/O设备之间巨大的速度差异。
  - 在大多数计算机上

        - 用户发出输入输出请求，由操作系统分配调度 I/O设备，并进行具体的I/O处理，使I/O与 CPU、主存的操作尽可能并行。
        - 对**应用程序员**透明
- 性能指标
    - 输入输出速度；
    - 用户从输入到输出的等待时间；
    - CPU和主存的利用率；
    - I/O系统的兼容能力
    - 可扩展能力，综合处理能力，性能价格比等。
- **误区：**使用多进程技术可以忽略I/O性 能对系统性能的影响。
    - 多进程技术只能够提高系统吞吐率，并 不能够减少系统响应时间；
- 特点
    - 异步性
    - 实时性
    - 与设备无关性
- **控制方式**
    - 程序控制方式（软件实现）
        - 特点：CPU直接进行I/O操作。
            - 何时、对何设备进行I/O操作完全受CPU控制；
            - CPU要通过指令对设备进行测试才能知道设备的工作状态。如空闲、准备就绪、正在忙 碌等；
            - 数据的**输入和输出都要经过CPU**；
            - 用于连接**低速外围设备**，如终端、打印机等。
        - 优点：灵活性很好；可以很容易地改变 各台外围设备的优先级。
        - 缺点：浪费CPU资源，速度慢。实现处 理机与外围设备并行工作困难。
    - 中断方式（软件实现）
        - 特点
            - 数据的输入和输出都要经过CPU ；
            - 使CPU与外围设备能够**并行**工作；
            - 能够处理例外事件。例如，电源掉电等；
            - 灵活性好； □用于连接低速外围设备。
    - DMA(直接存贮器访问)方式 （硬件实现）
        - 特点
            - 在主存与外设之间有直接访问通路；
            - CPU挪用一个存储周期启动DMA操作；
            - 在DMA方式开始和结束时，需要处理机进行管理；
            - 在DMA方式的数据传送过程不需要CPU干预。
        - 优点：输入输出与CPU并行工作，提高 了速度和效率
        - 缺点：外设的管理、DAM启动、数据准 备、操作完毕后的处理还需由CPU完成
        - 主要用来连接**高速外围设备**，例如磁盘
    - I/O处理机方式（硬件实现）

    - 特点
        - 把对外围设备的输入输出和管理工作从 CPU分离出来，由专门的处理机完成；
        - 实现CPU与I/O设备操作的并行。
    - 方式
        - 通道（或通道处理机）方式
        - 外围处理机方式

3.2 磁盘阵列（RAID）

- 定义：将一组磁盘驱动器用某种逻辑方式联系 起来，作为逻辑上的一个磁盘驱动器来 使用。
- 优点
    - 成本低，功耗小，传输速率高。
    - 提供容错功能，安全性更高。
    - 同样容量下，价格要低许多。
- 背?

RAID 级别	名称	最少数据磁盘数	可正常工作的最多失效磁盘数	检测磁盘数
RAID0	无冗余无校验的磁盘阵列	2	0	0
RAID1	镜象磁盘阵列	2	1	0
RAID2	纠错海明码磁盘阵列	2	1	1
RAID3	位交叉奇偶校验的磁盘阵列	2	1	1
RAID4	块交叉奇偶校验的磁盘阵列	2	1	1
RAID5	无独立校验盘的奇偶校验磁盘阵列	2	1	1
RAID6	双维无独立校验盘的奇偶校验磁盘阵列	2	2	2

3.3 总线

- 定义：总线是一组能为多个部件**分时共享**的公 共信息传送线路。
    - **共享**是指总线上可以挂接多个部件， 各个部件之间相互交换的信息都可以 通过这组公共线路传送；
    - **分时**是指同一时刻总线上只能传送一 个部件发送的信息。
- 优点：成本低；简单；易用
- 缺点：总线的带宽形成了信息交换的**瓶颈**，从而限制了系统中总的I/O吞吐量。
- 总线事务：通常把在总线上一对设备之间的一次信息交换过程 称为一个“总线事务”。
- 性能指标
    - 总线宽度 ： 总线的线数，它决定了总线 所占的物理空间和成本。 （地址线 & 数据线）
    - 总线带宽：总线的大数据传输速率，即 每秒传输的字节数。

    - 总线负载：指连接在总线上的大设备数量。 大多数总线的负载能力是有限的。
    - 总线复用：指在不同时段利用总线上同 一个信号线传送不同信号，例如地址总线和 数据总线共用一组信号线。
    - 总线猝发传输：在一个总线周期中可以传输存储地址连续的多个数据。
- 设备使用总线的步骤
    - 总线请求、总线仲裁、寻址、传送数据、检错和报错。
- 总线传输技术问题
    - 总线传输同步。为使信息正确传送，防止丢失，需对总线 通信进行定时。
    - 总线仲裁控制。在总线上某一时刻只能有一个总线主部件 控制总线，为避免多个部件同时发送信息到总线的矛盾， 需要有总线仲裁机构。
    - 出错处理。数据传送过程中可能产生错误，有些部件有自 动纠错电路，可以自动纠正错误。有些部件虽无自动纠错 能力，但能发现错误，这时可发出“数据出错”信号，通 知CPU来进行处理。
    - 总线驱动。通常采用三态输出电路或集电极开路输出电路 来驱动总线。
- 总线定时控制
    - 同步：系统采用一个统一的时钟信号来协调发送和 接收双方的传送定时关系。
    - 异步：完全依靠传送双方相互制约 的“握手”信号来实现定时控制。

3.4 通道处理机

- 目标：进一步减少CPU的传送控制工作负担。
- **通道定义**：具有特殊功能的处理器，执行程序实现 I/O传送控制。
- 作用
    - 减轻CPU处理输入输出的负担；
    - 高效管理设备，使设备并行工作，提高利用率；
- 特点
    - 拥有通道指令和通道程序
    - 可与cpu并行工作
    - 由通道统一管理外设
- 过程
    - **传送准备**。在用户程序中使用访管指令进入管理程序，由CPU通过管理程序组织一个通道程序，并启动通道。
    - **数据传送**。通道处理机执行通道程序，完成指定的数据输入输出工作。
    - **传送结束**。通道程序结束后第二次调用管理程序对输入输出请求进行处理。
- 分类
    - 字节多路通道
    - 数组多路通道
    - 选择多路通道
- 通道流量分析
    - 通道流量：在数据传送期间，单位时间内传送的字节数
        - 通道极限流量：通道所能达到的大通道流量
        - 实际流量：通道上挂接设备后所有设备要求的通道流量
    - 字节多路通道流量：分时为多台低速和中速外设服务的，每选择一台设备只传送一个字节。

        - P台设备每台传送n个数据共需时间$T_{byte}=(T_s+T_d)\times P\times n$
        - 极限流量$f_{max}=\frac{P\times n}{(T_s+T_d)\times P\times n}=\frac{1}{T_s+T_d}$
        - 实际流量：p台设备传输率之和
    - 选择通道流量：在一段时间内只能单独为一台高速外设服务， 当这台设备的数据传送工作全部完成后，通 道才能为另一台设备服务
        - $T_{select}=(\frac{T_s}{n}+T_D)\times P\times n$
        - $f_{max}=\frac{p\times n}{(\frac{T_s}{n}+T_D)\times P\times n}=\frac{n}{T_S+nT_D}$
        - 实际流量：p台设备中传输率最高者
    - 数组多路通道流量：每连接一台高速设备，就传送一个定长数 据块。传送完成后，又与另一台高速设备连接，再传送一个数据块
        - $T_{BLOCK}=(\frac{T_S}{k}+T_D)\times P\times n$
        - $f_{max}=\frac{p\times n}{(\frac{T_S}{k}+T_D)\times P\times n}=\frac{k}{T_S+kT_D}$
        - 实际流量：p台设备中传输率最高者
- 通道设计的基本原则
    - 通道的实际流量不超过通道的极限流量。
    - 一般安排传送速率高的设备请求具 有较高的响应优先级。

第四章 存贮体系

4.1 存贮体系及其性能

- 定义：两个或两个以上速度、容量和价格各不相同 的存储器用硬件、软件、或软件与硬件相结 合的方法连接起来成为一个存储系统。
    - 虚拟存储器：主存 - 辅存 存贮层次 ——解决容量问题（软件 & 硬件）
    - Cache和主存 存贮层次 —— 解决速度问题（硬件only）
- 两个原则
    - 一致性
    - 包容
- 程序的局部性 —— 预判的基础
    - 时间局部性——最近的未来要用到的信息可能就是当前正在使用 的信息——这是由程序的**循环**造成的。
    - 空间局部性——近的未来要用到的信息可能就是当前信息的相 邻信息——这是由程序的**顺序执行**造成的
    - 预判的准确性是存贮层次设计好坏的主要标志。
- 性能计算
    - 以两级存储M_1, M_2为例，T_{A_i}为CPU访问到M_i中的信息所需要的时间
    - H为CPU产生的地址能在M_1中访问到的概率
    - 等效访问时间T_A
        - 若二者同时启动

          $T_A=HT_{A_1}+(1-H)T_{A_2}$
        - 若M1不命中才启动M2

$$T_A = HT_{A_1} + (1-H)(T_{A_1} + T_{A_2}) = T_{A_1} + (1-H)T_{A_2}$$

- 访问效率$e = \frac{T_{A_1}}{T_A} = \frac{T_{A_1}}{HT_{A_1} + (1-H)T_{A_2}} = \frac{1}{H + (1-H)r}$，其中$r = \frac{T_{A_2}}{T_{A_2}}$

- 改进：预取 —— 在不命中时，把M2中相邻的几个单元组成的数据块都取出来送入M1
  - 预取后的命中率$H' = \frac{H + n - 1}{n}$，n为数据块大小*数据重复使用次数（可以理解为不命中率降低n倍）

4.2 虚拟贮存器

- 透明性：对应用程序员透明，对系统程序员不透明
- 管理方式
  - 段式
  - 页式
  - 段页式
- 段式管理
  - 基本思想：将程序按逻辑意义分成段，按段进 行调入、调出和管理。
- 页式虚拟存贮器
  - 地址映像与变换
    - 定义：将每个虚存单元按某种规则（算法）装入 （定位于）实存，即建立多用户虚地址Ns 与实主存地址np之间的对应关系
    - 全相联地址映像
      - 每道程序的任何一个虚页都可以装入到主存的 任何一个实页位置。 仅当同时调入主存的虚页数超过2^nv（实页数） 时，才会出现实页冲突。
      - 方法
        - 页表法
        - 相联目录法
          - 将页表压缩，只存放已装入主存的那些虚页与 实页位置的对应关系。该压缩了的页表多有 2^nv行
  - 页面替换算法
    - 随机RAND —— 命中率低但是易于实现
    - 先进先出FIFO
    - 近期最少使用LFU
    - 近期最久未使用LRU
      - 检测方法
        - 随机期法
          - 一旦使用位都为1，由硬件强制全部使用位为0。这段时 间是随机的
        - 定期法
          - 定期地使全部使用位为0 ， 为每个实页配“未使用过计数器”Hs

        - 优先替换算法OPT ： 在时刻t 找出主存中每个页将要用到的时刻 ti。 选择ti-t 大的页面进行替换。 —— 不现实的，作为评价算法
        - 堆栈型替换算法
            - 定义：对任意一个程序的页地址流作两次主 存页面数分配，分别分配m个主存页面和 n个主存页面，并且有m≤n。如果在任何 时刻t，主存页面数集合Bt都满足关系： Bt(m) <= Bt(n) 则这类算法称为堆栈型替换算法
            - 堆栈型替换算法的**命中率随分配的实页数的增加而单调上升**，至少不会下降。
            - **LRU和OPT是堆栈型，FIFO不是**
    - 提高虚拟存贮器性能的措施
        - 提高主存命中率
            - 程序地址流、页面调度策略、替换算法、页面 大小、以及分配给程序的实页数（主存容量）等都会影响主存命中率。
            - 两个存储器的速度不要相差太大
            - 加快内部地址变换
                - 相联目录表
                - 增加快表
                - 散列

4.3 高速缓冲存贮器Cache

- 基本结构和工作原理
    - 读取：将CPU给出的主存地址 变换为Cache地址，搜索Cache
        - 未命中：找主存
        - 命中：返回
    - 写入：将CPU给出的主存地址 变换为Cache地址，搜索Cache
        - 未命中：写主存
        - 命中：写Cache，写主存 —— 一致性问题
    - 特点
        - Cache与CPU采用相同工艺；
        - 地址映象、变换、替换、调度等由专门的硬件实现；
        - Cache靠近CPU或就放在CPU中，以减少与CPU之间 的传输延迟；
        - Cache—主存之间的信息交换对所有程序员都透明；
        - Cache访问主存的优先级高于其他系统访问主存的优 先级；
        - 除了Cache和CPU有直接的通路外，主存和CPU也有 直接的通路，可以实现读直达和写直达
    - 地址映像规则
        - 把Cache和主存等分成相同大小的块， 这样，Cache—主存地址映像就演变为主存中 的块如何与Cache中的块相对应。
        - 方法
            - 全相联映像与变换：主存中的任意一块都可以装入到 Cache中的任意一个块位置。
                - 地址变换：采用相联存储器构成的**目录表**，以 硬件方式实现
            - 直接映像与变换 ： 主存中的每一块只能装入到Cache内唯一一 个指定的块位置。

        - 地址变换：模Cache块数做映射 二进制意义下可以直接取低位
        - 优点
            - 所需硬件简单，成本较低；
            - 访问Cache可与访问区号表、比较区号等操作同时进行，节省了地址变换时间。
        - 缺点
            - 块冲突概率很高；
            - Cache空间利用率很低。因此已经很少使用。
    - 组相联映像与变换
        - 映像算法
            - 将整个Cache看成是一个区，将Cache分成G 组（G=2^g），每个组S块（S=2 ^ s）
            - 将主存分成2e个与Cache大小相同的区，每个 区分成G组（G=2 ^ g），每个组S块（S=2 ^ s）
            - 组间直接映像 组内全相联映像
- Cache替换算法
    - 随机
    - FIFO
    - LRU
- Cache替换算法的实现
    - 堆栈法： 使用硬堆栈
    - 比较对法：使用逻辑电路、触发器
        - 让各个块成对组合，用一个触发器的状态来 表示该比较对内两块访问的远近次序，再经 门电路就可找到LRU块。
        - $$C_{LRU} = T_{AB} \cdot T_{AC} \cdot T_{BC} + \overline{T_{AB}} \cdot T_{AC} \cdot T_{BC} = T_{AC} \cdot T_{BC}$$
          $$B_{LRU} = T_{AB} \cdot \overline{T_{BC}}$$
          $$A_{LRU} = \overline{T_{AB}} \cdot \overline{T_{AC}}$$
- Cache 透明性问题
    - 问题：写Cache时如何更新主 存的内容
    - 方法
        - 写直达法：利用直接通路，把 数据同时写入Cache和主存 （多处理机）
        - 写回法：为每个Cache块设置“修改位”，仅当替换时，才把修改过的Cache块写回到 主存（单处理机）
- 基本Cache优化方法
    - 降低失效率
        - 增加Cache块的大小
        - 增大Cache容量
        - 提高相联度
    - 减少失效开销

        - 多级Cache
        - 使读失效优先于写失效
    - 缩短命中时间
        - 避免在索引缓存期间进行地址转换

第五章 流水线和向量处理机

5.1 重叠方式

- 重叠原理：重叠解释执行两条或多条指令，加快程 序的执行
- 执行方式（认为一条指令分取址、分析、执行三部分）
    - 顺序解释执行方式：指令以及其微指令都顺序解释执行。
    - 一次重叠执行方式
        - 每次只重叠执行两条指令
        - $T=(1+2n)t$
    - 二次重叠执行方式
        - 每次重叠执行三条指令，第i条的分析执行第i+1条的取址，第i条的执行 执行第i+2条的取址。
        - $T=(2+n)t$
    - 重叠执行面临的问题以及解决
        - 功能部件冲突 ——执行、分析的重叠
            - 花费一定的硬件代价，分别设置独立的取指部 件、指令分析部件、指令执行部件；
            - 三个子过程都有独立的控制器：存储控制器、 指令控制器、运算控制器。
        - 访存冲突 —— 取址、分析的重叠
            - 采用两个独立编址、可同时访问的的 存贮器，分别存放操作数和指令。
            - 维持操作数和指令的混存在主存中， 但设置两个Cache： 指令Cache、数据Cache（哈佛结构）
            - .维持操作数和指令的混存，采用多体交叉存 贮器，采取低位交叉存取方式，让第k条指令 的操作数和第k+1条指令存放在不同的存贮体 。但这种方法有一定的局限性，不能从根本上 解决冲突。
            - 在主存和指令分析部件之间增设FIFO 的指令缓冲寄存器（又被称为先行指令 缓冲站）。利用主存的空闲，预先将下 一条或下几条指令取入指缓，而不必访 问主存储器。这样就能够使取指令、分 析指令和执行指令重叠起来执行。
        - 同步——“执行”和“分析”所需要的时间常常不完 全相同，指令分析部件和指令执行部件经常 要相互等待，从而造成功能部件的浪费
            - 采取先行控制技术 ——缓冲 技术和预处理技术，以及这两者的相结合。
        - 因此在重叠方式的机器中，应尽可能不使用条 件转移指令。 若使用应尽可能使用**不成功的转移指令**。

5.2 流水方式

- 流水线技术：将指令的执行过程分解为多个子过程，并让每个子过程分别由**专用的部件**完成，这些功能部件可以同时工作。让多条指令在时间上错开，依次通过各功能部件，这样，每个子过程就可以与其他的子过程并行进行，从而实现多条指令的并行执行， 减少多条指令或一段程序的完成时间。
- 特点
  - 基于**时间重叠**的并行处理技术
  - 不能加快一 条指令的执行，能加快多条指令的执行
  - 由多个有联系的子过程组成。这些子过程称为流水线的“级”或“段”
  - 段的数目称为流水线的“深度”
  - 每个子过程分别由专用的部件完成。这些部件可以同时工作
  - 各流水段的时间应尽量相等
  - 流水线工作阶段可分为**建立**、**满载**和**排空**三个阶段：
    - 从第一个任务进入流水线到流水线所有部件都处于工作状态这个时期，称为流水线建立阶段；
    - 当所有部件都处于工作状态时，称为流水线满载阶段；
    - 从最后一条指令流入流水线到结果流出，称为流水线的排空阶段。
    - 建立和排空阶段所用的时间分别叫做“装入时间”和“排空时间”。
  - 适合大量重复的时序过程。只有连续不断地向流水线输入任务，才能发挥流水线的效力。
  - 一般指令流水线有4-12个流水段。 等于及大于8个流水段的处理机称为超流水线处理机。
- 分类
  - 部件级、处理机级、系统级
  - 单功能、多功能
  - 静态、动态
  - 线性、非线性（有无前馈 / 后馈）
  - 向量、标量
  - 同步、异步
- 流水线的主要性能指标
  - 吞吐率：流水线单位时间内能处理的指令 条数或能输出的结果数。
    - $TP = \frac{n}{T_k}$ n为任务数，T_k为处理完成n个任务所用的时间
    - 最大吞吐率
      - 若各段时间相等，$TP_{max} = \frac{1}{\Delta_0}$
      - 若各段时间不相等，$TP_{max} = \frac{1}{max\{\Delta t_i\}}$受限于最慢子过程所用的时间
  - 加速比
    - $S_p = \frac{T_{非流水}}{T_{流水}}$
    - 各段不相等时：$T_{流水} = (n-1)max(\Delta t_1 \dots \Delta t_m) + \sum t_i$
- 效率：指流水线中的设备实际使用时间占整个运行时间之比，也称为流水线的 设备时间利用率
- 流水线调度
  - 线性流水线调度
    - 无反馈，只需每隔Δt（或数个Δt）输入一个任务即可。
  - 非线性流水线调度

        - 看书吧……
- 流水线相关
    - 局部相关：由于机器同时解释执行多条指令 时，这些指令对同一存贮单元要求“先 写后读”而造成的。
        - 同步流动
        - 异步流动
    - 全局相关：由分支转移指令引起的相关
        - 条件转移分支判断方法
            - 猜测法
            - 加快和提前形成条件码
            - 采用延迟转移
            - 加快短循环程序的处理

5.3 向量流水机

- 定义：将向量数据表示与流水处理方式结合在一 起，构成向量流水处理机，也称其为向量处理 机
- 水平处理
- 垂直处理
    - 采用多体交叉存贮。
    - 设置向量寄存器组。
        - 寄存器组内垂直，组间水平
- 结构
    - 存储器 - 存储器结构
    - 寄存器 - 寄存器结构

5.4 超级处理机

- 定义
    - 超标量处理机
        - 在不同的流水线中 执行不相关指令的能力。
    - 超流水线处理机
        - 一个周期内能够分时发射多条指令的处理机 称为超流水线处理机。
        - 指令流水线有8个或更多功能段的流水线处 理机称为超流水线处理机。
- 提高处理机性能的方法
    - 超标量处理机利用资源重复，设置多个 执行部件寄存器端口
    - 超流水线处理机则侧重开发时间并行性 ， 在公共硬件上采用较短的时钟周期、 深度流水来提高速度
- 性能比较
    - 最高：超标量处理机； 其次：超标量超流水线处理机； 最低：超流水线处理机。
- 原因
    - 超流水线处理机的启动延迟比超标量处理机大。
    - 条件转移造成的损失，超流水线处理机要比超标量 处理机大。

- 超标量处理机指令执行部件的冲突要比超流水线处 理机小。

第六章 并行处理机与互联网络

6.1 并行处理机原理

- 定义：重复设置多个同样的处理单元（PE）， 并将它们按照一定方式互相连接，在统 一的控制部件（CU）作用下，各自对分 配的数据并行地完成同一条指令所规定 的操作，实现操作级并行的SIMD（单指令多数据流）计算 机。（也称阵列处理机）
- 特点
  - 指令串行执行
  - 数据并行处理
  - 速度快，模块性好，可靠性高，效率低（vs流水线/向量机）
  - 依赖于互连网络和并行算法。
  - 需要有一台高性能的标量处理机。
- 基本构型
  - 分布式存贮器的阵列处理机
  - 集中式共享存贮器的阵列处理机
- SIMD和向量机的区别
  - 在向量计算机中，同一条指令可以连续地处理一个数据组， 而这些数据是以流水线的方式通过处理机，即采用一种时间复用的方式，
  - 在SIMD计算机中，通过大量处理单元对向量中包含的各分 量同时进行处理，也就是说，SIMD是通过硬件资源的重复设置来实现并行运算的，即采用空间复用方式。

6.2 互联网络（ICN）

- 定义：互连网络是一种由开关元 件按照一定的拓扑结构和控制方式构成的网 络，用来实现计算机系统内部多个处理机或 多个功能部件之间的相互连接。
- 主要特性
  - 网络规模：网络中结点的个数
  - 结点度：与结点相连接的边数称为结点度。包括入度和出度。进入结点的边数叫入度，从结点出来的边数则叫出度 结点度= 入度+ 出度
  - 距离：两个结点之间相连的最少边数
  - 网络直径：网络中任意两个结点间距离的最大值。用结点间的连接边数表示
  - 结点间的线长：两个结点之间连线的长 度。用米、公里等表示
  - 等分宽度：某一网络被切成相等的两半 时，沿切口的最小边数称为该网络的等分 宽度
  - 对称性：从任何结点看到拓扑结构都是 一样的网络称为对称网络。对称网络比较 易实现，编程也较容易
- 分类
  - 静态互连网络
  - 循环互连网络
  - 多级互连网络
  - 全排列互连网络
  - 全交叉开关网络
  - 动态互连网络

- 单级互联网络
  - 立方体
  - PM2I
  - 混洗交换
- 多级互联网络
  - 多级立方体
  - 多级PM2I
  - 多级混洗交换

第七章 多处理机与多计算机

7.1 多处理机定义

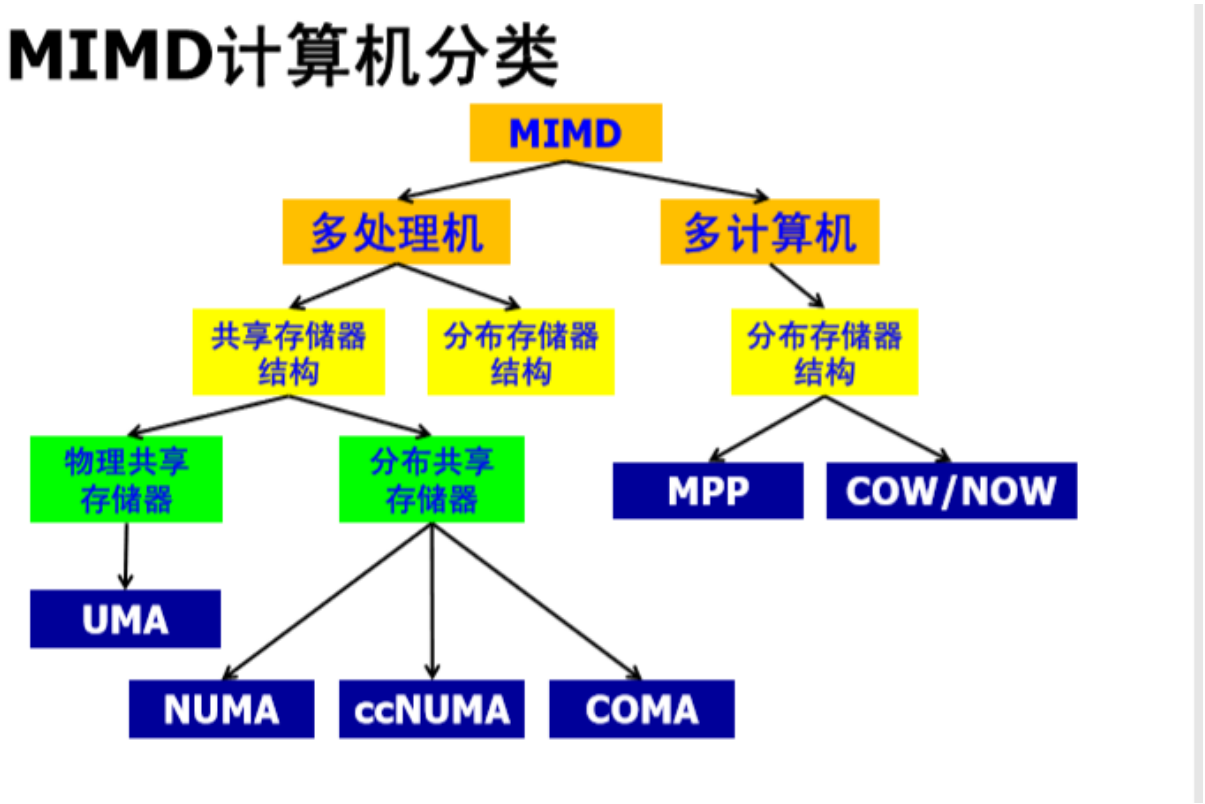
- 由两台及以上处理机组成的计算机系统； 各处理机有自己的控制部件、局部存储器，能执行各自的程序，可以共享公共主存和所有外设； 各处理机通过某种形式互连，相互通信； 实现作业、任务、指令、数据等各个级别的并行； 属于**MIMD**
- 优点
  - 提高性能；
  - 提高可靠性；
  - 减少及其功耗；
  - 提高效费比；
- 分类
  - 实现并行技术的途径
    - 同构型——资源重复
      - 处理机类型/功能相同
      - 并行处理或执行任务
    - 异构型——时间重叠
      - 处理机类型/功能不同
      - 重叠处理各任务
    - 分布式——资源共享
      - 处理机类型/功能相同或 不同
      - 并行/协同完成任务处理
      - 由统一操作系统统一管理 资源
  - 耦合度：多处理机之间物理连接的紧密程度和交叉作 用的能力的强弱
    - 紧耦合多处理机
      - 直接耦合系统
      - 通过公共硬件（如存储器，I/O系统等）通信
      - 典型：SMP（对称多处理机），多核处理器等
      - 一般将紧耦合多处理机称为多处理机
    - 松耦合多处理机
      - 间接耦合系统

        - 通过通道、通信线路或网络、消息传递系统通信
        - 典型：机群，计算机网络等
        - 一般将松耦合多处理机称为多计算机

- 特点
    - 结构灵活：满足多种并行计算的不同需求
    - 程序并行：实现作业、任务之间的并行
    - 并行任务派生：可并行执行任务的识别、派生与 分配
    - 进程同步：解决数据相关和控制依赖问题
    - 资源调度与分配：动态分配资源和调度任务，以获得更好性能和更高效率

7.2 多处理机结构

- 共享存储器结构
    - 特点
        - 各处理机共享存储器，并通过对共享存储器 读/写实现相互通信；
        - 对存储单元的任何修改对其他处理机都是可 见的；
        - 延迟低，但扩展性差；
    - 实现方式
        - UMA
            - 均衡存储器访问结构或集中式共享存储器结构
            - SMP 对称多处理机
        - NUMA
            - 非均衡存储器访问结构 或 分布式共享存储器
            - ccNUMA：高速缓存一致性非均匀存储访问 ， 各处理机cache内容一致
        - COMA
            - 仅用高速缓存存储器结构
            - 将NUMA中的分布式存储器换成了cache
- 分布式存储器结构
    - 非远程存储访问模型
    - 特点
        - 各处理机拥有自己的本地存储器，在本地操作系统 控制下独立工作。
        - 各处理机的本地存储器是私有的，不能被其他处理 机访问。
        - 各处理机借助互连网络、通过消息传递机制相互通 信，实现数据共享。
        - 大规模并行处理机（MPP）、机群（cluster）等采 用了这种结构
    - 优点
        - 结构灵活，扩展性较好
    - 缺点
        - 任务传输以及任务分配算法复杂，通常要设 计专有算法
        - 处理机之间的访问延迟较大
        - 需要高带宽的互连



7.3 多核处理器

- 是多处理机的一种特殊形式：SMP on a single chip
- 定义：一枚处理器中集成了两个或多个独立处理单 元（称为核）的处理器；
- 优点
  - 提高吞吐率和并行程序的速度
  - 可以实现核的紧耦合
  - 更好地实现核之间的通信（相比SMP）
  - 共享Cache
  - 降低功耗
  - 降低时钟频率
  - 可以挂起空闲的核
- 缺点
  - 仅能提高并行程序的速度
  - 扩大了与存储器速度的差距

7.4 多处理机多cache一致性

- 顺序一致性：存储器访问次序同程序执 行次序一致
- 不一致的原因
  - 共享可写数据
  - 进程迁移
  - I/O传输（DMA）
- 解决方案

- 硬件
  - Cache一致性协议
    - 监听协议：拥有数据副本的Cache各自记录数据块的 状态，无集中的状态。
    - 基于目录的协议 ： 将每个数据块的共享状态 记录保存在某个地点—目录
- 软件
  - 由编译和操作系统检测并解决

7.5 多处理机操作系统

- 分类
  - 主从式(Master-slave)
    - 由一台主处理机进行系统的集中控制，负责记录、 控制其它从处理机的状态，并分配任务给从处理机。
    - 优点：硬件和软件结构相对简单
    - 缺点：对主处理机可靠性要求很高。当不可恢复 错误发生时，系统容易崩溃，此时必须重新启动主处理机。系统灵活性差，在控制使用系统资源 方面效率也不高。
  - 独立监督式(Separate Supervisor)
    - 操作系统将控制功能分散给多台处理机，共同完 成对整个系统的控制工作。每个处理机均有各自的管理程序(操作系统的内核)。
    - 优点
      - 每个处理机都有其专用的管理程序，故访问公用表格 的冲突较少，阻塞情况自然也就较少，系统的效率较 高
      - 每个处理相对独立，因此一台处理机出现故障不会引 起整个系统崩溃
    - 缺点
      - 减少了对控制专用处理机的需求，但是实现更复杂
      - 每个管理程序都有一套自用表格，但仍有一些共享表 格，从而发生表格访问冲突问题，导致进程调度复杂 性和开销的加大
      - 修复故障造成的损害或重新执行故障机未完成的工作 非常困难
      - 各处理机负荷的平衡比较困难。
  - 浮动监督式(Floating Supervisor)
    - 系统中每次只有一台处理机作为执行全面管理功 能的“主处理机”，“主处理机”可以根据需要 浮动，即从一台切换到另一台处理机。这样，即 使执行管理功能的主处理机故障，系统也能照样 运行下去
    - 优点
      - 系统可靠性更强，没有单主处理崩溃瓶颈
      - 更好的平衡处理机负载
    - 缺点：实现复杂