```yaml
title: 大学课程 | 计算机系统结构
tags:
  - 系统结构
  - 大学课程
categories:
  - 学习笔记
abbrlink: 50754
reward: true
copyright: true
date: 2020-04-13 10:33:59
cover: https://npm.elemecdn.com/justlovesmile-img/1584111180-soft.jpg
top_img: https://npm.elemecdn.com/justlovesmile-img/1584111180-soft.jpg
```

作者博客：Justlovesmile's BLOG

大三计算机系统结构知识点总结笔记

计算机系统结构

第一章 计算机系统结构基础及并行性的开发

- 计算机性能的高速增长受益于
  - 电路技术的发展
  - 体系结构技术的发展
  - 其他因素(OS, Compiler 的发展)
- 80年代后，RISC技术和微处理器技术使得体系结构技术对计算机性能发展的影响越来越大

> CISC(复杂指令集)结构出现得较早。在这种结构中，新功能的增加主要靠增加新的指令，但为了保持向下兼容，就必须保持原有的指令；每条指令都有不同的操作指令码，对应不同的数据类型和位置，这样就造成了系统具有较大的指令系统和复杂的寻址技术，指令格式也很不规范。而RISC(精简指令集)结构则采用定长指令，使用流水线的方式执行指令。这种结构大量利用寄存器间的操作，大大简化了处理器的结构，优化了VLSI器件的使用效率，同时功耗较低。与CISC处理器相比，RISC处理器的突出特点是只用硬件实现最常用的指令，其他指令通过微代码软件来模拟实现，通过简短的定长指令提高并行度。这样虽然硬件设计较简单，但处理器指令的逻辑设计反而复杂了。RISC诞生之时恰逢386处理器取得了巨大的成功。在当时条件下，同样工艺水平的芯片，采用RISC架构的产品，其速度比CISC快3倍左右，这在Intel内部产生了很大的震动。最终诞生的 486处理器是第一个真正引入RISC技术的X86处理器。

1.1 计算机系统的层次结构

- 如何从整体上认识计算机系统?
  - 一种新的认识方法：从计算机语言的角度，将计算机系统看成按**功能**划分的**多级层次结构**

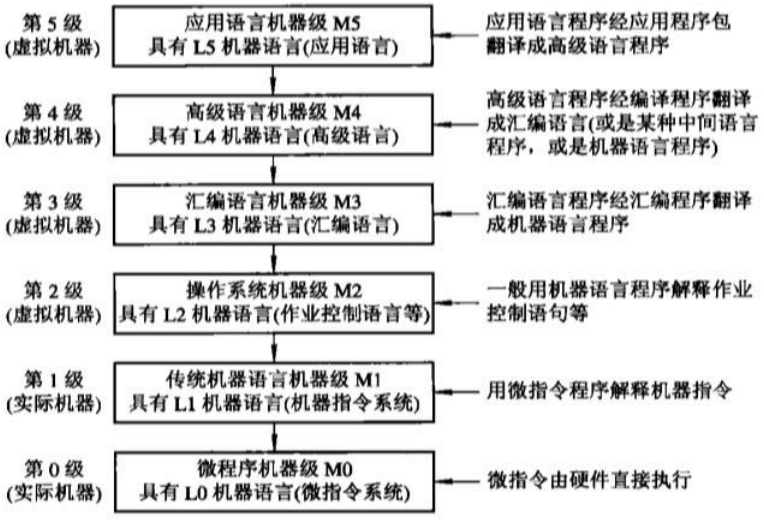
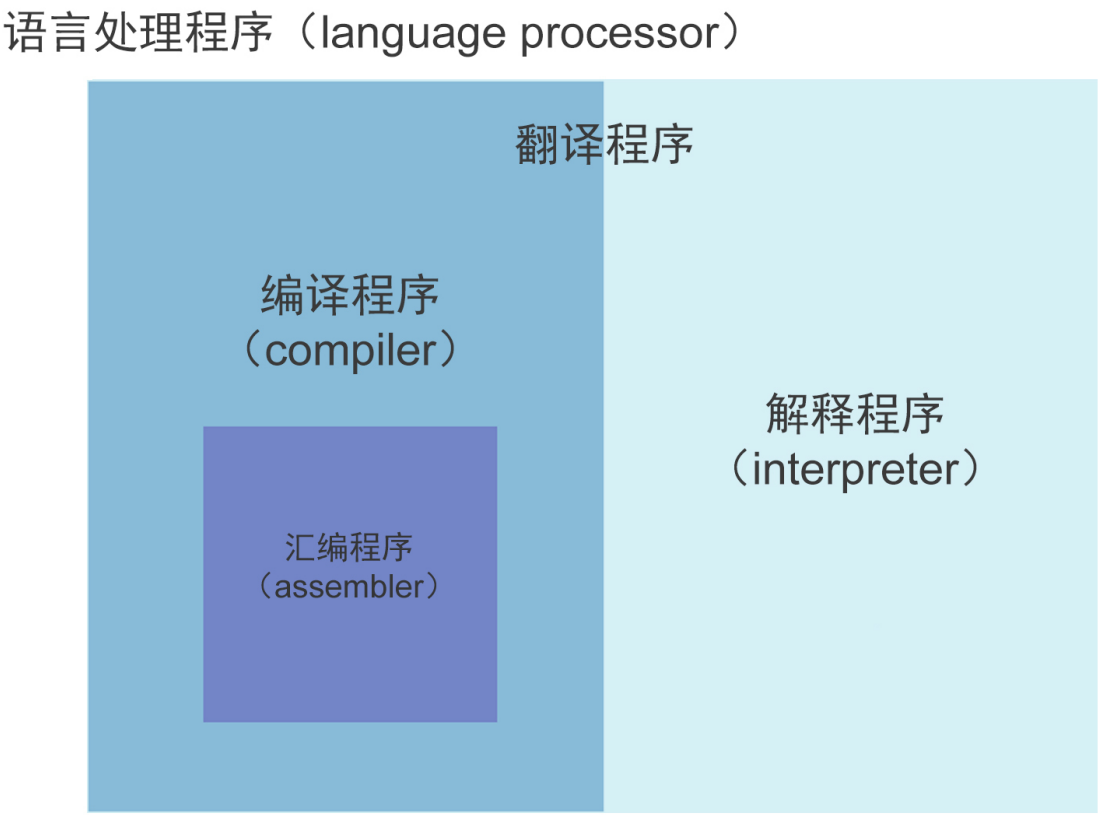


图 1－1　计算机系统的多级层次结构

- M0用硬件实现，M1用微程序（固件）实现，M2到M5大多用软件实现
- 固件：是一种具有软件功能的硬件
- 虚拟机：由软件实现的机器。虚拟机的功能不一定全由软件实现，也可以是固件或硬件
- **选择什么样的软硬件比例**，是系统结构研究的核心内容之一
- 多层系统结构的意义和作用
  - 推动了计算机系统结构的发展
    - 有利于正确理解软件、硬件和固件在系统结构中的地位和作用
  - 发展了多处理机系统、分布处理系统、计算机网络系统等系统结构
    - 每级有各自的用户、实现方法和指令集，摆脱各级功能在一台机器实现
  - 推动自虚拟机、多种操作系统共行等技术
- 对于使用某一级语言编程的程序员来说，只需要熟悉和遵守该级语言的使用规定。
- 各机器级的实现采用翻译技术或解释技术，或者两者结合
- **翻译**：先用转换程序将高一级机器级上的程序**整个**地变换成低一级机器级上等效的程序，然后再在低一级机器级上实现的技术

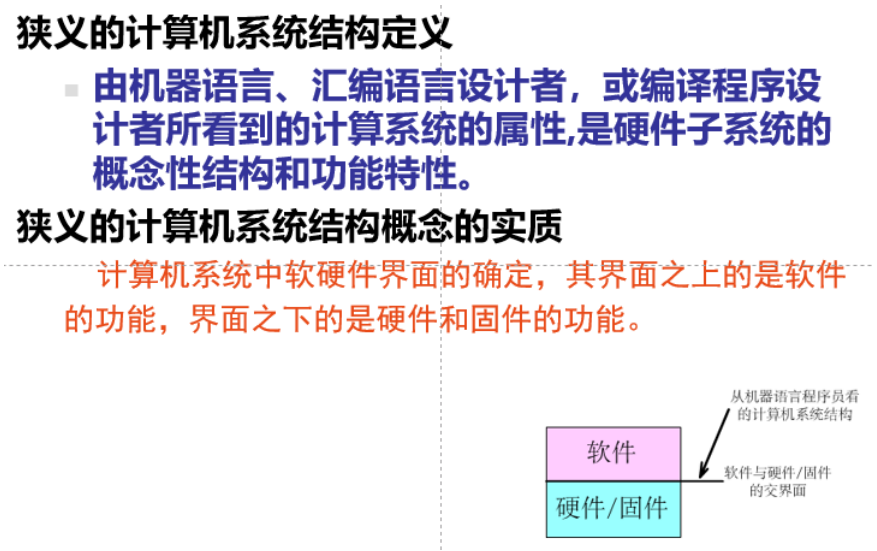
- **解释**：在低级机器级上用它的一串语句或指令来仿真高级机器级上的一条语句或指令的功能，是通过对高级的机器级语言程序中的每条语句或指令**逐条**解释来实现的技术



1.2 计算机系统结构，计算机组成和计算机实现

1.2.1 计算机系统结构的定义和内涵

从计算机系统的层次结构角度来看，系统结构是对计算机系统中各级界面的定义及其上下的功能分配。计算机系统的每一级都有自己的系统结构。



- 从不同级看到的计算机属性是不同的
- 计算机系统的层次结构具有的特征：**透明性**
- 透明：客观存在的事物或属性从某个角度**看不到**
  - 优点：可以不用管理它，简化设计
  - 缺点：看不到而无法加以控制，会带来不利

从汇编语言程序员的角度来看，以下哪些是透明的?

指令地址寄存器	否
指令缓冲器	是
时标发生器	是
条件码寄存器（状态寄存器）	否
乘法器	是
主存地址寄存器	是
磁盘外设	否
先行进位链	是
移位器	是

计算机系统结构也称为计算机系统的体系结构，它只是系统结构中的一部分，指的是传统机器级的系统结构。

- 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
- 计算机系统结构是程序员所看到的计算机的属性，即概念性结构与功能特性

就目前的通用机来说，计算机系统结构的属性包括：

（1）硬件能直接识别和处理的数据类型及格式等的数据表示。

（2）最小可寻址单位、寻址种类、地址计算等的寻址方式。

（3）通用/专用寄存器的设置、数量、字长、使用约定等的寄存器组织。

（4）二进制或汇编指令的操作类型、格式、排序方式、控制机构等的指令系统。

（5）主存的最小编址单位、编址方式、容量、最大可编址空间等的存储系统组织。

（6）中断的分类与分级、中断处理程序功能及入口地址等的中断机构。

（7）系统机器级的管态和用户态的定义与切换。

（8）输入/输出(I/O)设备的连接、使用方式、流量、操作结束、出错指示等的机器级I/O结构。

（9）系统各部分的信息保护方式和保护机构等属性。

【例 1－1】 IBM PC 系列和 VAX－11 系列的指令系统、寻址方式、寄存器组织、I/O设备连接方式等都不一样，从传统机器语言程序员或汇编语言程序员角度看，概念性结构和功能特性差异很大。要使他们所编的程序能运行，应了解的计算机属性大不相同，但高级语言程序员却看不到这些差异。

> 堆栈型机器、累加器型机器和通用寄存器型机器各自有什么优缺点
> 1.堆栈型机器——CPU 中存储操作数的单元是堆栈的机器。
> 2.累加型机器——CPU 中存储操作数的单元是累加器的机器。
> 3.通用寄存器型机器——CPU 中存储操作数的单元是通用寄存器的机器。
> CPU状态分为管态和目态，CPU的状态属于程序状态字PSW的一位，管态又称特权状态、系统态或核心态。通常，操作系统在管态下运行，CPU在管态下可以执行指令系统的全集。
> 目态又称常态或用户态，机器处于目态时，程序只能执行非特权指令。用户程序只能在目态下运行。

1.2.2 计算机组成和计算机实现的定义及内涵

1.**计算机组成**：是指**计算机系统结构的逻辑实现**，包括机器级内部的数据流和控制流的组成以及逻辑设计等等
（1）着眼点：机器内部各事件的排序方式与控制机构，各部件的功能及各部件间的联系。
（2）预解决：在合理或满足要求的性能和价格的条件下，怎么最佳、最合理地把各种设备和部件组织成计算机，以实现所确定的计算机。

- 计算机组成设计要确定的方面包括：
  - 数据通路宽度（数据总线上一次并行传送的信息位数）
  - 各种操作对功能部件的共享程度
  - 专用功能部件的设置
  - 功能部件的并行性
  - 缓冲和排队技术
  - 预测技术
  - 可靠性技术
  - 控制机构的组成，等等

2.**计算机实现**：是计算机组成的**物理实现**
（1）着眼点：器件技术（主导作用），微组装技术

- 计算机实现:(是数字电路等课程主要研究的内容)

- 处理器、主存的物理结构
- 器件的集成度和速度
- 信号传输
- 器件、模块、插件、底板的划分与连接
- 涉及的专用器件
- 电源、冷却、微组装技术、整机装配技术，等等

【例 1－2】 指令系统的确定属于计算机系统结构研究的范畴。指令的实现，如取指令、指令操作码译码、计算操作数地址、取数、运算、输出结果等的操作安排和排序属于计算机组成研究的范畴。实现这些指令功能的具体电路、器件的设计及装配技术属于计算机实现研究的范畴。

以指令系统、乘法指令系统及主存系统为例，说明系统结构、组成和实现各自所研究的内容

	系统结构	组成	实现
指令系统	指令系统的确定	指令的实现，如取指，译码，求AE取操作数等设计与排序	实现各指令的电路设计、器件的设计及装配
乘法指令	确定是否有乘法指令	指令是由加法器连加实现还是用专门的乘法器实现	加法器或乘法器的类型、集成度、数量、价格、组装技术
主存系统	主存的容量与编址方式	主存的速度Tm、单体多字或多体多字等的逻辑结构	器件的选定、电路的设计、微组装技术（双极型、MOS型）

- 系列机：CPU的机器指令和汇编指令系统相同（或绝大部分相同）
- **狭义的系统结构**是**组成**的抽象
- **组成**是**实现**的抽象
- 一种体系结构可以有多种组成，一种组成可以有多种物理实现
- **计算机系统结构**研究的范畴：机器/汇编指令系统，数据表示，是否采用通道方式输入/输出的确定
- **计算机组成研究**的范畴：指令采用顺序，重叠，流水还是其他方式解释，数据通路宽度的确定，通道采用结合型还是独立型
- **广义**的计算机体系结构定义：包括**狭义的系统结构，计算机组成**
  - 任务：
    - 从程序设计者角度：软硬件的功能分配以及确定软硬件界面
    - 从计算机设计者角度：更合理地实现分配给硬件的功能

1.2.3 计算机系统结构，组成和实现的相互影响

- 计算机系统结构，组成，实现三者互不相同，但又互相影响
- 不同的系统结构会影响到组成技术
- 组成技术也会影响系统结构，是一种推动作用
  - 专用部件的设置
- **实现**永远是结构和组成的最坚实基础

1.3 计算机系统的软硬件取舍，性能评测及定量设计原理

1.3.1 软，硬件取舍的基本原则

- 计算机系统结构的任务：
  - 软、硬功能分配，确定软硬界面

提高硬件功能比例	提高软件功能比例
提高解题速度	降低解题速度
减少程序所需存储空间	增加程序所需存储空间
增加硬件成本	减少硬件成本
降低软件费用	增加软件费用
降低硬件利用率	增加系统灵活性
降低系统灵活性	

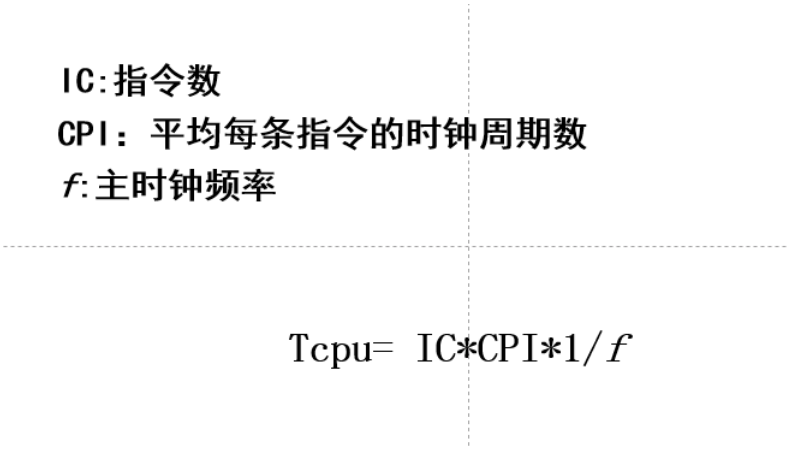
- 软硬件取舍的**三原则**：
  - 在现有的硬件和器件条件下，系统要有**高的性能价格比**；
  - 在软硬功能分配时，要考虑到准备采用的组成和实现技术，**使其不过多的限制或不合理限制各种组成、实现技术**；
  - 在软硬功能分配时，除了从“硬件”角度考虑，还要从“软”的角度考虑，把为**编译、OS以及高级语言的设计与实现提供更多、更好的硬件支持放在首位。**

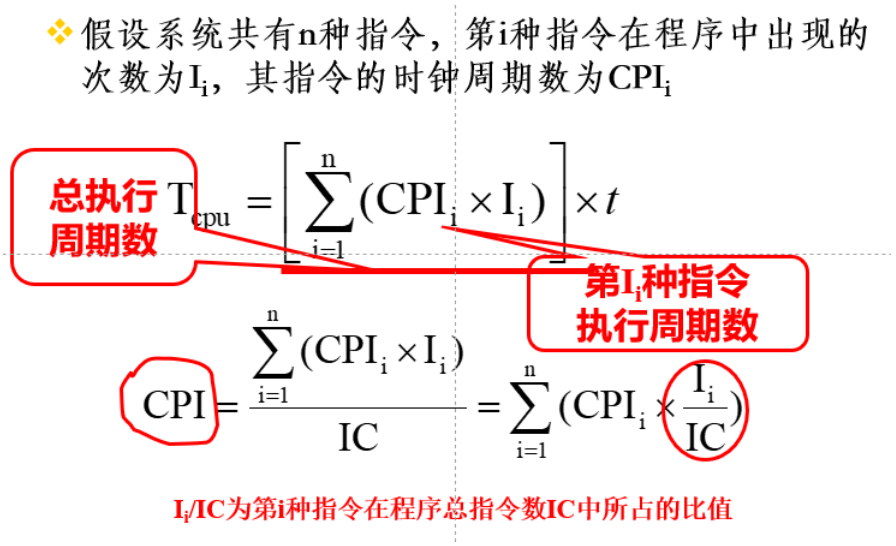
- 语义差距的大小实质上取决于软硬件功能的分配

1.3.2 计算机系统的性能评测与定量设计原理

1.计算机系统的性能评测

- 时钟频率：CPU的主频表示在CPU内数字脉冲信号震荡的速度，与CPU实际的运算能力并没有直接关系
- 计算机系统的性能指标体现于**时间**和**空间**两个方面，在系统上程序实际运行的**时间**应该是衡量机器时间（速度）性能最可靠的标准
- 机器的性能是通过采用好的硬件，系统结构以及高效的资源管理等技术来提高的
- 计算机性能指：数据处理（数据运算（速度），数据传输（速度）），数据容量，数据质量的综合性能
- 系统性能的测量依赖于人（计算机用户/系统管理者）的观点
- 计算机的性能以及对系统评价的目标都指**系统速度的性能**，通常是用**响应时间**来衡量





- 减少CPI是RISC思想的精华

- 反应程序的运行速度通常引入下面一些指标
  - MIPS：计算机单位时间执行的指令条数
    - 主频越高f，平均每条指令的时钟周期数CPI越少，其MIPS越高，在一定程度反映了机器的性能
    - MIPS很大程度的依赖指令集，它很难衡量指令系统不同机器之间的性能
    - 用于比较相同指令系统系统
    - 即使在同一台机器上，程序负荷不同，CPI也不同，MIPS也就受到影响——浮点运算、定点运算
    - MIPS还与机器的硬件实现有关
      - 浮点运算在硬件上实现，MIPS低，性能高
      - 浮点运算用软件实现，MIPS高，性能低
  - MFLOPS：每秒百万次浮点运算
    - Tflops:每秒一万亿次浮点运算
    - 1T=1024G， 1G=1024M

- 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：理想情况下计算机系统可以获得的最高理论性能值
  - 持续性能：也称**实际性能**，其表示有**算术性能平均值**，**调和性能平均值**，**几何性能平均值**

- 算术性能平均值
  - 设算术性能平均值Am是n道程序运算速度或运算时间的算术平均值
  - 以速率评价

$$A_m = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{T_i} = \frac{1}{n}\left(\frac{1}{T_1} + \frac{1}{T_2} + \ldots + \frac{1}{T_n}\right)$$

- Ti是第i个程序的执行时间，Ri是第i个程序的执行速率
- 以执行时间评价

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

- 调和性能平均值
  - 调和平均数又称倒数平均数，**给定数据的倒数之算术平均数的倒数**
  - **Hm的值与运行全部程序所需的时间成反比---比较准确**
  - m

$$H_m = \frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}} = \frac{n}{\sum_{i=1}^{n} T_i} = \frac{n}{T_1 + T_2 + \cdots + T_n}$$

- 几何性能平均值
  - 几何平均数是n个变量值连乘积的n次方根
  - $$G_m = \sqrt[n]{\left(\prod_{i=1}^{n} R_i\right)} = \sqrt[n]{\left(\prod_{i=1}^{n} \frac{1}{T_i}\right)}$$
  - $$\frac{G_m(X_i)}{G_m(Y_i)} = G_m\left(\frac{X_i}{Y_i}\right)$$

    可以进行机器性能对比
  - 几何平均值无法给出系统性能的真实期望.
  - 几何平均值常常使用测试机和参考机之间归一化的比值
- **调和平均数、几何平均数和算术平均数三者间，存在如下数量关系：H≤G≤A**

❖ 考虑工作负荷中各程序出现的次数，可将程序出现的执行速率加权

- **加权算术平均值**

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

- **加权调和平均值**

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

- **加权几何平均值**

$$G_m = \prod_{i=1}^{n} (R_i)^{\alpha_i} = R_1^{\alpha_1} \times R_2^{\alpha_2} \times \ldots \times R_n^{\alpha_n}$$

2.计算机系统的定量设计原理

（1）哈夫曼压缩原理

- 也称关注经常性事件原则
- 抓主要矛盾
- 性能——功耗的折中
  - 处理器执行两个数的加法运算，溢出不溢出情况
  - Pentium M处理器为了降低系统功耗且同时提高计算机性能，在其译码单元引入“micro-op Fusion”概念，把原有的两个micro-op（microinstructions）合成为一个进行操作

（2）Amdahl定律

- 该定律将“关注经常性事件原则”进行了量化

- 用于确定对系统中**性能瓶颈部件采取措施提高速度后**能得到系统性能改进的程度，即系统加速比Sp
- **Sp定义为系统改进后的性能与未改进时的性能的比值，或者定义为系统未改进时的程序执行时间Told与改进后程序执行时间Tnew的比值**
- Sp与两个因素有关，即性能可改进比fnew和部件加速比rnew
- 性能可改进比fnew是系统性能可改进部件占用的时间与未改进时系统总执行时间的比值
  - 0≤fnew≤1
- 部件加速比rnew是系统性能可改进部分在改进后性能提高的比值
  - Rnew>1
  - Rnew=TRold/TRnew
-

$$S_p = \frac{\text{没有采用改进措施前执行某任务的时间}}{\text{采用改进措施后执行某任务的时间}} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{(1-f_{new})*T_{old}+T_{rnew}} = \frac{T_{old}}{(1-f_{new})*T_{old}+T_{rold}/r_{new}} = \frac{T_{old}}{(1-f_{new})*T_{old}+T_{old}*f_{new}/r_{new}} = \frac{1}{(1-f_{new})+f_{new}/r_{new}}$$

- 当fnew为0时，Sp=1；当Rnew趋于无穷大时，Sp=1/(1-fnew);
- 通过使用某种较快的执行方式所获得的性能提高，受限于该部件占用系统执行时间的百分比,它是一个悲观定律
- 通用多核系统在扩展到8个核以上时，往往会达到一个性能降低的拐点
- 应用本身受到串行处理模式和技术的限制。无论拥有多少个核心，它们中的许多都因等待数据进行串行处理而被闲置起来
- 阿姆达尔定律描述的一个关键事实是它**只适用于计算的一种场合**，即施行并行化后计算中的顺序部分将占据执行时间的主要部分
- 阿姆达尔定律是在**固定应用规模的前提下**考虑并行性增长的影响。但大多数并行计算则是**固定并行性而扩展应用的规模**

（3）程序访问的局部性规律

- 时间局部性: 一个存储项被访问,可能很快再访问
- 空间局部性: 存储项被访问,它的邻近项可能很快被访问

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：f_{new}=0.4, r_{new}=10，根据Amdahl定律

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

某一计算机用于商业外贸的事务处理，有大量的字符串操作。由于这种事务处理很普遍，有较大的市场，故而设计人员决定在下一代此类计算机的CPU中加入字符串操作的功能。经测试应用软件调查发现，字符串操作的使用占整个程序运行时间的50%，而增加此功能如用软件（如微程序）实现，则快5倍，增加CPU成本1/5倍；如果用硬件实现，则快100倍，CPU成本增加到5倍。问设计人员提出增加此功能是否恰当？是否用软件还是硬件？（设CPU成本占整机成本的1/3）

❖ **硬件实现**

$$T_{new} = T_{old}(1-50\%) + \frac{50\%\,T_{old}}{100}$$

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-50\%)+\frac{50\%\,T_{old}}{100}} = \frac{1}{1-50\%+\frac{50\%}{100}} = 1.98\ \text{倍}$$

$$\text{成本增加} = \frac{2}{3}*1+\frac{1}{3}*5 = 2.33\ \text{倍}$$

$$\text{成本性能比} = \frac{2.33}{1.98} = 1.18\ \text{倍}$$

❖ 软件实现

$$T_{new} = T_{old}(1-50\%) + \frac{50\%\,T_{old}}{5}$$

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-50\%) + \frac{50\%\,T_{old}}{5}} = \frac{1}{1-50\% + \frac{50\%}{5}} = 1.66\ \text{倍}$$

$$\text{成本增加} = \frac{2}{3} * 1 + \frac{1}{3} * (1 + \frac{1}{5}) = 1.07\ \text{倍}$$

$$\text{成本性能比} = \frac{1.07}{1.66} = 0.64\ \text{倍}$$

1.3.3 计算机系统设计的主要任务和方法

1.计算机系统设计的主要任务

- 包括系统结构，计算机组成和计算机实现的设计

  2.计算机系统的设计方法

- 由上往下
  - 先考虑如何满足用户要求，定好面对使用者的虚拟机的基本功能和环境，如指令系统、语言结构、数据类型等，然后逐级向下设计，每一级都严格考虑优化上一级来实现
  - 

- 由下往上
  - 不考虑用户的具体需求，只根据当时拿到的器件，参照或吸收已有各种机器的特点，把微程序机器级和传统机器级分别研制出来，然后再配上适合不同应用环境的各种操作系统和各种编译程序，以满足不同方面的应用要求。
  - 优点:
    - 有利于缩短研制周期
    - 有利于软硬件人员之间的交流
    - 软硬分配更加合理
    - 系统的性能价格比更高
- 从中间往两头

1.4 软件，应用，器件的发展对系统结构的影响

1.4.1 软件的发展对系统结构的影响

- 软件的可移植：软件不用修改或只需经少量加工就能由一台机器投到另一台机器上运行，即同一软件可以应用于不同的环境
- 软件移植的三种技术
  - 统一高级语言：面向题目和算法，和具体结构关系不大
    - 指一种完全通用的高级语言，为所有程序员所使用，并能在完全不同的机器之间实现程序的软件移植。
  - 系列机、兼容机：具有相同结构的各种机器之间
    - 兼容机：不同厂家生产的具有相同指令集结构的计算机
    - 系列机：CPU的机器指令和汇编指令系统相同（或绝大部分相同）
    - 软件兼容：即同一个软件可以不加修改地运行于体系结构相同的各档机器，而且它们 所获得的结果一样，差别只在于有不同的运行时间。
    - 向上(下)兼容指的是按某档机器编制的程序，不加修改的就能运行于比它高(低)档的机器。
    - 向前(后）兼容指的是按某个时期投入市场的某种型号机器编制的程序，不加修改地就能运行于在它之后(前)投入市场的机器。
    - 系列机软件兼容性必须保证做到向后兼容，力争向上兼容
  - 模拟与仿真：软件在不同体系结构之间的移植
    - 模拟
      - 用A机的机器语言程序解释B机器语言程序，从而实现软件移植方法 宿主机 A机 虚拟机 B机
      - 为实现模拟编制的各种解释程序

- 仿真
  - 用A机的微程序解释B机的机器语言程序，从而实现软件移植方法 宿主机 A机 目标机 B机
  - 为实现仿真编制的各种解释微程序
-

	模拟	仿真
A机	宿主机	宿主机
B机	虚拟机	目标机
解释程序	模拟程序	仿真微程序
存放介质	主存	控存
实现	复杂	简单
速度	慢	快
灵活性	大	小
适用范围	运行时间短，实时性要求不高	两个机器的结构差异不大
解释次数	双重解释	单重解释，但有二套解释程序

三种软件移植方法比较

方 法	适应范围	要 求	其 他
统一高级语言	系统结构相同或不同	语言标准化	
系列机	系统结构相同	软件兼容	兼容阻碍了系统结构的突破性进展
模拟	相同或不同	结构差异过大，效率太低	灵活性最大
仿真	结构差异不太大	具有微程序机器级	

1.4.2 应用对系统结构的影响

- 应用需求是促使计算机系统结构发展的最根本的动力
- 不同的应用对计算机系统结构的设计提出了不同的要求
- 计算机应用可归纳为：数据处理，信息处理，知识处理，智能处理

1.4.3 器件的发展对系统结构的影响

- 芯片制造的发展
- 器件的发展对系统的影响
  - 功能和使用方法—非用户片、现场片及用户片
  - 改变了逻辑设计的传统方法—速度、规整
  - 推动了系统结构技术的发展
  - 体系结构"下移"速度加快—并行计算
  - 促进了算法、语言和软件的发展
- 软件是促使计算机系统结构发展的最重要的因素
- 应用需求是促使计算机系统结构发展的最根本的动力
- 器件是促使计算机系统结构发展最活跃的因素
- 非用户片：也称通用片，其功能是由器件厂家生产时已确定的，器件的用户（即机器设计者）只能使用，不能改变器件内部功能
- 现场片：是用户根据需要可改变器件内部功能的芯片（FPGA）
- 用户片：是专门按用户要求生产的高集成度VLSI器件（ASIC）
  - 全用户片：是完全按用户要求设计的用户片
  - 半用户片：是基本按通用片进行生产，最后按用户要求再制作的用户片，如门阵列、门-触发器阵列等

1.5 系统结构钟的并发性开发及计算机系统的分类

1.5.1 并行性的概念和开发

1.并行性的含义和级别

- **并行性**：解题中具有可以同时进行运算或操作的特性

- 并行性包含了同时性和并发性二重含义
  - **同时性**（Simultaneity）：两个或多个事件在**同一时刻**发生
  - **并发性**（Concurrency）：两个或多个事件在**同一时间间隔内**发生

- **只要时间上有重叠就存在并行性！**

- 并行性的等级：
  - 从计算机系统中执行程序的角度（由低到高）
  - 指令内部—一条指令内部各个微操作之间的并行
  - 指令之间—多条指令的并行执行
  - 任务或进程之间—多个任务或程序段的并行执行
  - 作业或程序之间—多个作业或多道程序的并行。
  - 从处理数据的角度（由低到高）
  - 位串字串——顺序
  - 位并字串——同时对一个字的全部位
  - 位片串字并——同时对许多字的同一位
  - 全并行——同时对许多字的全部或部分
  - 从计算机信息加工的各个步骤和阶段
  - 存储器操作并行
  - 处理器操作步骤并行
  - 处理器操作并行
  - 指令、任务、作业并行

  2.并行性开发的途径

- **时间重叠**
  - 是在并行性概念中引入时间因素，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。
    - 举例：流水线

- **资源重叠**
  - 是在并行性概念中引入空间因素，通过重复设置硬件资源来提高可靠性或性能
    - 例如：双工系统

- **资源共享**
  - 是利用**软件的方法**让多个用户按一定时间顺序轮流地使用同一套资源，以提高其利用率，这样也可以提高整个系统的性能
  - 例如：网络打印机
  - 多道程序、分时OS →真正的处理机代替虚拟机→分布处理系统

  3.计算机系统的并行性发展

- 不同时间阶段，并行性发展的主要表现不同

  4.多机系统的耦合度

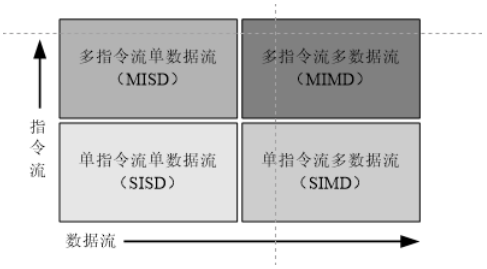
- 多机系统：包含多处理机系统和多计算机系统
  - 多处理机系统
    - 是由多台处理机组成的单一计算机系统，各处理机都可有自己的控制部件，可带自己的局部存储器，能执行各自的程序
    - **在逻辑上受统一的操作系统控制**，体系结构可以是共享存储器，也可以是分布式存储器
  - 多计算机系统
    - 是由多台独立的计算机组成的系统，**各计算机分别在逻辑上独立的操作系统控制下运行**，机间可以互不通信，即使通信也只是经通道或通信线路以文件或数据集形式进行，实现多个作业的并行
    - 一般指分布式存储结构
    - 集群系统和大规模并行处理机MPP都是多计算机系统

- **耦合度**：一般用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叉作用能力的强弱
  - **最低耦合系统**（Least Coupled System）：各种脱机系统
  - **松散耦和系统**（Loosely Coupled System）：如果多台计算机通过通道或通信线路实现互连，共享某些磁带、磁盘等外围设备，以较低频带在文件或数据集一级相互作用。又称间接耦合系统
  - **紧密耦合系统**（Tightly Coupled System）：如果多台机器之间通过总线或高速开关互连，**共享主存**，并有较高的信息传输速度，可以实现数据集一级、任务级、作业级的并行。又称直接耦合系统

1.5.2 计算机系统的分类

1.弗林分类

- 弗林分类法（Michael J,Flynn分类）：弗林提出按**指令流**和**数据流**的多倍性对计算机系统进行分类
  - **指令流**：是指机器执行的**指令序列**
  - **数据流**：是指指令流调用的**数据序列**，包括输入数据和中间结果
  - **多倍性**：是指在系统性能瓶颈部件上处于**同一执行阶段**的指令或数据的最大可能个数



❖ SISD 单指令流单数据流
- 传统的单处理机属于SISD计算机
- 单功能部件处理机：IBM1401，VAX-11
- 多功能部件处理机：IBM360/91，370/168

❖ SIMD 单指令流多数据流
- 并行处理机是SIMD计算机的典型代表
- 我国的YH-I型是此类计算机型
- 全并行：IILIAC IV、PEPE、STAR100、ASC、CRAY
- 字并位串：STARAN、MPP、DAP

❖ MISD 多指令流单数据流
- 目前看不存在，但也有学者认为存在

❖ MIMD 多指令流多数据流
- 包括了大多数多处理机及多计算机系统
- 我国的YH-II型计算机是这种类型的计算机
- 紧密偶合：IBM3081、IBM3084、UNIVAC-1100/80
- 松散偶合：D-825、Cmmp、CRAY-2

- 主要缺点：
  - (1)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
  - (2)根本问题是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
  - (3)非冯计算机的分类；其他新型计算机的分类

2.库克分类

- 用**指令流**和**执行流**（Execution Stream）及其多倍性来描述计算机系统总控制器的结构特征
  - SISE：单处理机系统
  - SIME：多操作部件的处理机
  - MISE：带指令级多道程序的单处理机
  - MIME：多处理机

- 缺点
  - 有些系统，如分布处理机等，没有总控制器
  - 分类太粗，如SIME中包含了多种类型的处理机

3.冯泽云分类

- 提出用数据处理的并行度来定量地描述各种计算机系统特性
  - WSBS（字串位串）
  - WSBP（字串位并）
  - WPBS（字并位串）
  - WPBP（字并位并）

- 缺点
  - 仅考虑了数据的并行性，没有考虑指令、任务、作业的并行

1. 按大小划分
   - 种类：巨型机、大型机、中型机、小型机、微型机等。
2. 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
3. 按数据类型划分
   - 种类：定点机、浮点机、向量机、堆栈机等
4. 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

1.6 本章小结

1. 重点：（1）计算机系统结构，计算机组成，计算机实现三者的定义以及包含的内容（2）有关透明性问题的判断（3）软件和硬件的功能分配原则（4）软件可移植性的途径，方法，适用场合，存在问题和对策（5）并行性的概念（6）系统结构中开发并行性的途径
2. 难点：透明性的判断与分析

> 下列哪些对**系统程序员**是透明的?
> **A、超大规模集成电路**
> B、虚拟存储器
> **C、Cache存储器**
> D、程序状态字
> E、“启动I/O”指令
> F、“执行”指令
> **G、指令缓冲寄存器**

正确答案： ACG

> 下列哪些对**应用程序员**是透明的?
> **A、虚拟存储器**
> **B、Cache存储器**
> **C、程序状态字**
> **D、“启动I/O”指令**
> E、“执行”指令
> **F、指令缓冲寄存器**

正确答案： ABCDF

> 1.系列机可将单总线改成双总线来减少公用总线的使用冲突。【答案：√】
> 2.系列机增加新机种时,为增加寻址灵活性和缩短平均指令字长,由原等长操作码改为多种码长的扩展操作码。【答案：×】
> 3.系列机应用软件应做到向前兼容,力争向下兼容。【答案：×】
> 4.可以说向后兼容是系列机的根本特征。【答案：√】
> 5.系列机不再是方向,因为它约束了计算机系统结构的发展。【答案：×】
> 6.由同一厂家生产的,系统结构相同的,但组成和实现不同的所有计算机,称为兼容机。【答案：×】

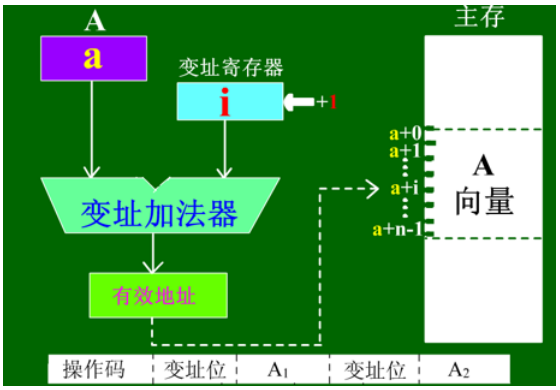
第二章 数据表示，寻址方式与指令系统

2.1 数据表示

2.1.1 数据表示与数据结构

- 数据表示：能由机器硬件识别和引用的**数据类型**，表现在它有对这种类型的数据进行操作的指令和运算部件。
- 数据类型：**不同于数据，数据类型除了指一组值的集合外，还定义了可作用于这个集合上的操作集**
  - **基本数据**类型
  - **结构数据**类型
    - 一组由相互有关的数据元素复合而成的数据类型，这些数据元素可以是基本数据类型中的元素，也可以是结构数据类型本身中的元素。也就是说这些数据是有结构的，**包括向量和数组、字符串、堆栈、队列、记录**等，结构数据类型中的元素不一定都具有相同类型
  - 访问指针
  - 抽象数据等类型
- 数据结构：通过软件映像，变换成机器中所具有的数据表示来实现的。
  - 是应用中相互之间存在一种或多种特定关系的数据元素的集合。如：线性表、栈、队列、串、数组、阵列、链表、树和图等。
  - 是结构数据类型的组织方式，它反映了结构数据类型中各种数据元素或信息单元之间的结构关系
  - 不同数据表示可以为数据结构的实现提供不同的支持。**数据表示是数据结构的子集**
  - 数据结构和数据表示是软硬件的交界面。

图为变址操作对向量，阵列数据结构的支持：



2.1.2 高级数据表示

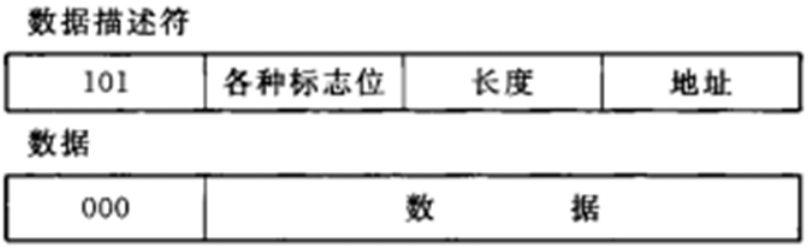
1.自定义数据表示
（1）标志符数据表示

- 为缩短高级语言与机器语言的语义差距，让机器中每个数据都带类型标志位
- | 类型标志 | 数据值 |
- 标志符数据表示的主要优点
  - 简化了指令系统和程序设计

        - 减少指令种类
    - 简化了编译程序
        - 不用做复杂的映射
    - 便于实现一致性校验
    - 能由硬件自动变换数据类型
    - 为软件调试和应用软件开发提供了支持
- 使用标志符数据表示可能带来如下问题
    - 每个数据字因增设标志符，会增加程序所占的主存空间
    - 采用标志符会降低指令的执行速度
        - 增加按标志符确定数据属性及判断操作数之间是否相容等操作，单条指令的执行速度会下降。
        - 程序的编制和调试时间的缩短，是解题总时间缩短
        - 所以，引入标志符数据表示对微观性能（机器的运算速度）不利，但对宏观性能（解题总时间）是有利的。

（2）数据描述符

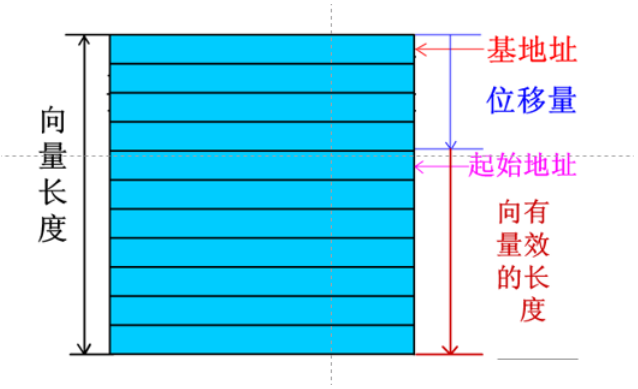
- 为了进一步减少标志符所占存储空间，对向量、数组、记录等数据，由于元素属性相同，因此就发展出数据描述符
- 描述符和数据分开，表示访问的数据是整块还是单个，地址信息等其他信息



- 描述符方法优点：
  - 描述符方法实现阵列数据的索引要比用变址方法实现更方便
  - 且便于检查出程序中的阵列越界错误
  - 数据描述符方法为向量、数组数据结构的实现提供了一定的支持，有利于并简化编译中的代码生成，可以比变址法更快地形成元素地址

2.向量数组数据表示

- 为向量、数组数据结构的**实现和快速运算提供更好的硬件支持的方法是增设向量、数组数据表示**
- 向量在内存中是连续存放在一段空间里的，换句话说，这些向量元素的地址是连续的
- 在标量计算机上运行时，由于没有专门的向量数据表示，因此在计算一个向量(相当于一维数组的计算)时，每取用一个数据元素，都要用到计算该元素的地址。
- 而在向量机中，由于有了向量数据表示，就**可以把一个向量用一个位串来表示出来。**向量指令就是能够用一条指令对向量的全部元素进行运算的指令。



- 引入向量、数组数据表示优点
  - 不只是能加快形成元素地址；
  - 便于实现把向量各元素成块预取到中央处理器；
  - 能对阵列中的每个元素又是一个子阵列的相关交叉型阵列进行处理；
  - 对稀疏矩阵能实现压缩存储、还原、运算等多种功能操作，不但节省了空间，也由于不必处理零元素而节省了时间。

3.堆栈数据表示

- 堆栈数据结构在编译和子程序调用中很有用
- 通用寄存器型机器对堆栈数据结构支持**较差**
    - 堆栈操作用的机器指令数少，功能单一，没有专门的堆栈指令
- 寄存器型机器的内存分配，有堆栈(Stack)空间
    - 堆栈置于存储器内，访问堆栈的速度低
    - 通常只用于保存子程序调用时的**返回地址**
    - 少量用堆栈来实现程序之前的**参数传递**

- 堆栈机器
  - 有堆栈数据表示的机器称为堆栈机器
  - 堆栈寻址方式的地址是隐含的，在指令中不必给出操作数的地址，**零地址指令**
  - 从60年代开始，出现了一批以堆栈寻址方式为主的堆栈计算机
  - 堆栈对以下这些方面处理非常方便
    - 表达式求值
    - 子程序调用，递归，中断嵌套
    - 块结构语言中的变量访问

堆栈计算机具有如下特点：
（1）有高速寄存器组成的硬件堆栈，并附加控制电路，让它与主存中的堆栈区在逻辑上构成整体，使堆栈的访问速度是寄存器级，容量是主存级。
（2）有丰富的堆栈指令且功能很强。
（3）支持高级语言，有利于编译程序。因为一般的算术表达式可以很容易地转化成逆波兰表达式，而逆波兰表达式能够直接形成由堆栈指令组成的程序，这样就简化了编译程序。

- **以主存寻址方式为主的计算机系统**，在编译一个算术表达式时，要**为每一个变量分配主存单元**，另外，还会人为地产生一些中间变量。**如何减少中间变量的个数**，合理地为变量分配存储单元，是编译器的一项许多相当困难的工作。
- **以寄存器寻址方式为主的计算机系统**，编译器需要决定哪些变量放在通用寄存器中，哪些变量放在主存中，**以减少访问主存储器的次数。**另外，也同样存在**如何减少了中间变量**，节省了存储空间的问题

（4）支持程序的嵌套和递归调用，支持中断处理

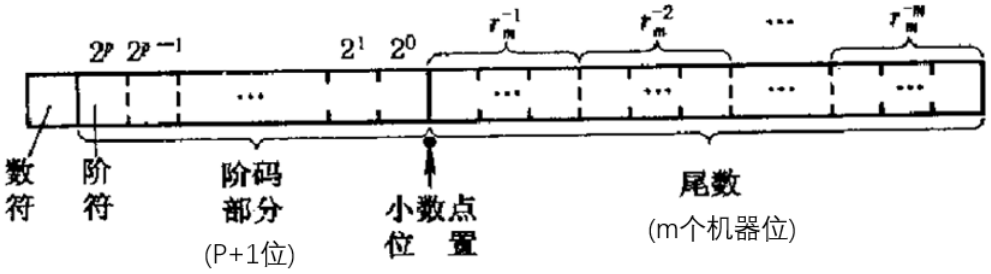
2.1.3 引入数据表示的原则

从根本上讲，存储器一维线性的存储结构与要求经常使用的多维离散数据结构有着很大的差距，不利于数据结构的实现。

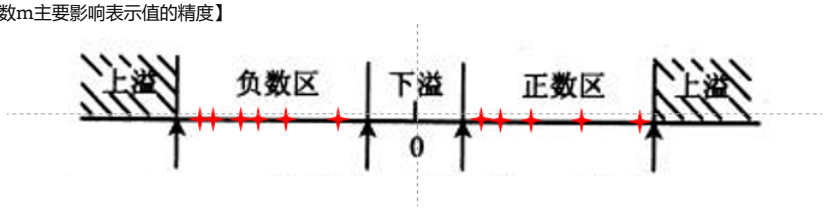
一是基本数据表示不可少；
二是看系统的效率是否显著提高；
三是看引入这种数据表示后，其通用性和利用率是否提高。
四是也需要挖掘基本数据表示的细节问题。

2.1.4 浮点数尾数基值大小和下溢处理方法的选择

1.浮点数尾数基值的选择
如果小数点的位置事先已有约定，不再改变，此类数称为“定点数”。
如果小数点的位置可变，则称为“浮点数”。



rm：尾数的基
re：阶码的基（re =2）
m：尾数长度 (注意其含义)
p：阶码长度
【p表示数的范围大小；尾数的位数m主要影响表示值的精度】



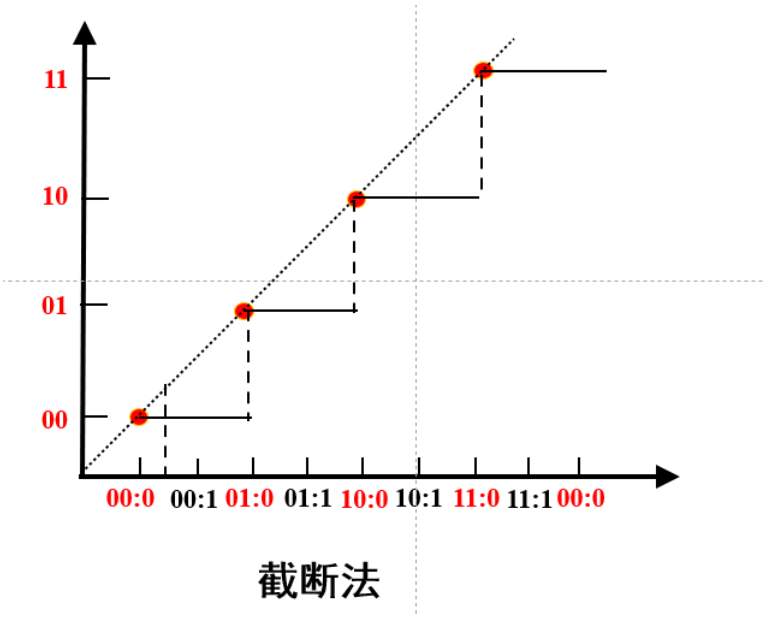
- 浮点数表数误差产生的原因：
  - 运算的结果
  - 十进制转化为二进制、四进制、八进制、十六进制
- 浮点数尾数基值的选择
  - 可表示数的范围
    - 随着rm增大，可表示的范围增大
    - 随着rm增大，可表示数的最小值rm^-1将减少
  - 可表示数的个数
    - (2^p)×(rm^m')×(1-(rm^(-1))，其中rm的增大将因`(1-(rm^-1))`增大而使可表示数的个数增多
  - 可表示的精度
    - rm越大，数在数轴上的分布越稀，数的表示精度自然就下降
  - 运算中的精度损失
    - 尾数右移出机器字长，使有效数字丢失，但其不同于可表示数的精度，由于尾数基值rm取大后，对阶移位的机会和次数减少，又由于数的表示范围扩大，使尾数溢出需右规的机会也减少。因此，rm越大，尾数右移的机会越小，精度的损失就越小
  - 运算速度
    - Rm增大时，由于对阶或尾数溢出需右移及规格化需左移的次数减少，运算速度可以提高

- 综上所述，尾数基值取大，会扩大浮点数的表示范围、增加可表示数的个数、减少移位次数、降低右移造成的精度损失和提高运算速度，但是会，降低数据的表示精度，数值的分布变稀
- 规格化正尾数：**正尾数小数点后第一个rm进制数位不是0的数**

条件：非负阶，规格化，正尾数	阶值：二进制p位，尾数：rm进制m'位	若p=2，m=4，当rm=2（即m'=4）时	若p=2，m=4，当rm=16（即m'=1）时
可表示最小尾数值	rm^(-1)	1/2	1/16
可表示最大尾数值	1-1×rm^(-m')	15/16	15/16
最大阶值	2^p-1	3	3
可表示最小值	rm^(-1)	1/2	1/16
可表示最大值	rm^(2^p-1)×(1-rm^(-m'))	7.5	3840
可表示的尾数个数	rm^(m')×(rm-1)/rm	8	15
可表示阶的个数	2^p	4	4
可表示数的个数	2^p×rm^(m')×(rm-1)/rm	32	60

2.浮点数位数的下溢处理方法
减少运算中的精度损失关键是要处理好运算中尾数超出字长的部分，使精度损失最小
（1）**截断法**
方法：**将尾数超出机器字长的部分去掉**

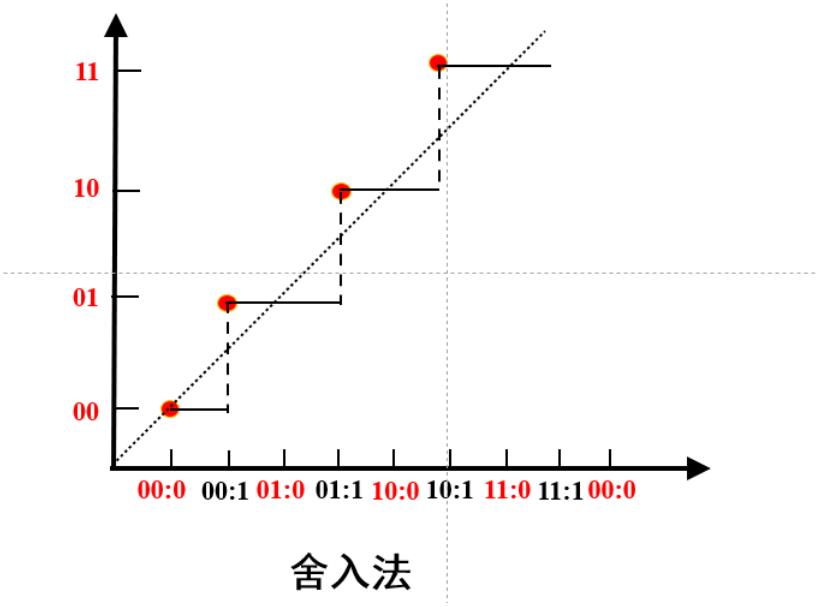
- 以rm=2，m=2为例讨论最大误差
  - 在整数时接近于1(“11:111…1”截断成“11:”)
  - 在分数时接近于2^(-m) (“.01:111…1”截断成“.01:”)
  总是产生负误差



- 优点
  - 实现简单，不增加硬件，不需要处理时间
- 缺点
  - 最大误差较大，且平均误差大且无法调节，因而已很少使用

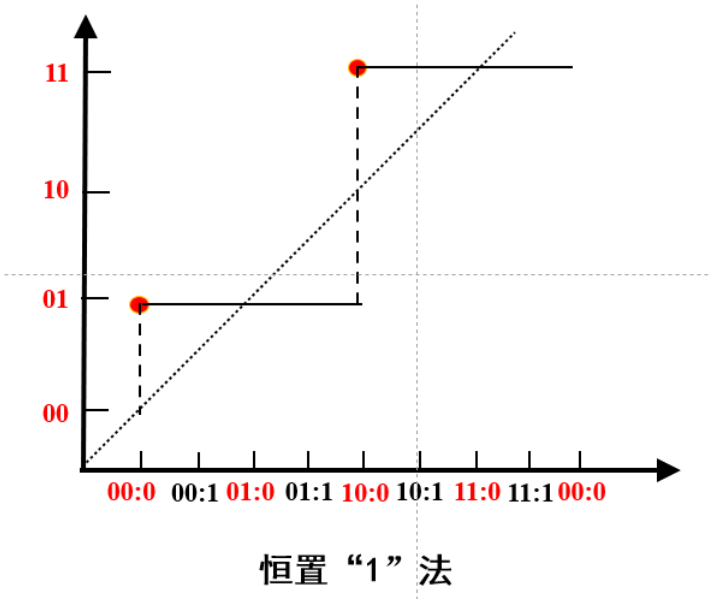
（2）**舍入法**
在机器运算的规定字长之外增设一位附加位，存放溢出部分的最高位，每当进行尾数下溢处理时，将附加位加1,(整数加0.5，分数加2-(m+1))

- 例如：
  - 整数：“10:10…0”舍入成“11:” 正误差
  - 分数：“.10:01…0”舍入成“.10:” 负误差
- 优点
  - 实现简单，增加的硬件开销少，最大误差小，平均误差接近于零
- 缺点
  - 处理速度慢，需要花费在数的附加位加1以及因此产生进位的时间，最坏情况下，需要从尾数最低位进制



（3）**恒置“1”法**

- 将机器运算的规定字长之最低位恒置“1”
- 最大误差
  - 整数为1（如“10:00…0”处理成“11:”）
  - 分数为2-m（如“.00:00…0”处理成“.01:”）
- 误差有正负
  - 负误差（如“.11:10…1”处理成“.11:”）
  - 正误差（如“.00:00…0”处理成“.01:”）

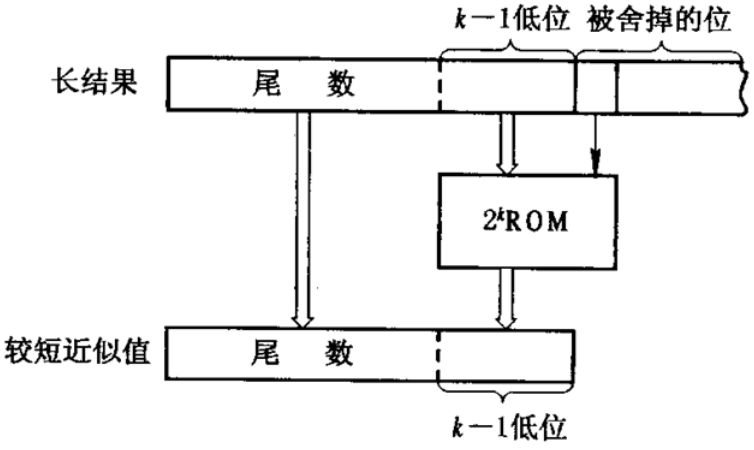


- 优点
  - 实现简单，不需要增加硬件和处里时间，平均误差趋于0
- 缺点
  - 最大误差最大，比截断法还大（接近于1）

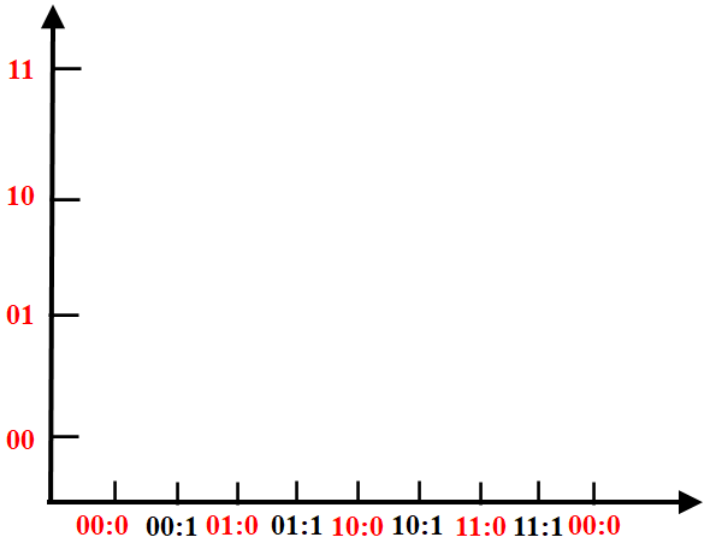
  多用于中、高速机器中，由于尾数位数比微、小型机器长

（4）查表舍入法

取尾数p位的最后k-1位和准备舍弃的最高1位，共k位。通过ROM或PLA查表得到k-1位，作为新的尾数p位的最后k-1位



- 下溢处理表的内容
  - 当尾数最低k-1位为全”1“时以截断法设置处理结果
  - 其余情况采用舍入法



- 优点
  - ROM法速度较快，平均误差可调到0
  - 避免再次右规操作
- 缺点
  - 需要硬件配合
- 上述4种处理方法中，
  - **最大误差最大**的是恒置“1”法，
  - **最大误差最小**的是舍入法；
  - **平均误差最大**的是截断法；
  - **平均误差可人为调节**的是查表舍入法；
  - **下溢处理不需要附加时间开销，即速度最快**的是截断法和恒置“1”法，
  - **处理速度最慢**的是舍入法；
  - **实现上最花费硬件**的是查表舍入法，
  - **最省硬件**的是截断法和恒置“1”法。

2.2 寻址方式

寻址方式指的是按什么方式寻找（或访问）到所需的操作数或信息的。

2.2.1 寻址方式的三种面向

面向主存，寻址主要访问主存，少量访问寄存器；
面向寄存器，主要访问寄存器，少量访问主存和堆栈；
面向堆栈，主要访问堆栈，少量访问主存或寄存器。

2.2.2 寻址方式在指令中的指明

- 具体的寻址方式，组成原理已经讲过
- 基本的指令格式：操作码 + 地址码
- 寻址方式指明方法：
  - 占用操作码的某些位指明
  - 不占操作码，在地址码设置寻址方式字段

2.2.3 程序在主存中的定位技术

- 逻辑地址与主存物理地址
  - 逻辑地址：程序员编写程序时使用的地址；
  - 物理地址：程序在主存中的实际地址；
- 早期——单道程序
  - 逻辑地址和物理地址是一致的，程序和数据存放在主存中的位置是由程序员编写程序时指明的；
- 现在——多道程序
  - 程序员已不用主存的实际地址编程，改用符号、标号名编址；
  - 由源程序中的符号名空间→目标程序的逻辑地址空间→主存中的物理地址空间
  - 程序员事先无法知道程序装在主存中什么位置；
  - 各道程序的逻辑地址都是从0开始编制，主存物理空间地址是从0开始编址的一堆线性空间；

1. 静态再定位
   （1）在目标程序装入内存时，由装入程序对目标程序中的指令和数据的地址进行修改，即把程序的逻辑地址都改成实际的内存地址。重定位在程序装入时一次完成
   （2）存在问题
   ①一道程序地址改错而导致其他程序出错；
   ②指令修改妨碍了程序的可重入。
   ③指令修改了，不利于程序定位和调试。
2. 动态再定位
   是在程序执行期间完成的，即程序的逻辑地址在装入内存时不作任何修改，程序执行中，每取出一条指令，CPU对其译码时，如果有逻辑地址，就借助于重定位机构将其转换成绝对地址，然后执行该指令。
3. 基址寻址
   （1）指令中给出一个形式地址（作为修改量），并给出基址寄存器号，基址寄存器内容（作为基准量）与形式地址相加得到操作数有效地址
   （2）主要解决
   ①程序重定位；
   ②扩展有限字长指令的寻址空间

4. 虚实地址映象表
   地址加界法要求程序员所用编址空间不能超出实际主存空间容量。

2.2.4 物理主存中信息的存储分布

目前使用最普遍的编址单位是字节编址，这是为了适应非数值计算的需要

2.3 指令系统的设计和优化

2.3.1 指令系统设计的基本原则

- 指令系统是软、硬件的主要界面
- 指令系统的设计主要包括**指令的功能**（操作类型、具体操作内容）和**指令格式**的设计.
- 指令设计的步骤：
  - **根据应用**，初拟出指令的分类和具体的指令；
  - 试编出用该指令系统设计的各种**高级语言的编译程序**；
  - **大量测试**程序进行**模拟测试**，看指令系统的操作码和寻址方式效能是否都比较高；
  - **将程序中高频出现的指令串复合改成一条强功能新指令**，即改用硬件方式实现；而将频度很低的指令的操作改成基本的指令组成的指令串来完成，即用软件方式实现；
- 系统设计人员希望：指令码密度适中，兼容性，适应性
- 指令的组成：
  - 一般的指令主要由两部分组成：**操作码**和**地址码**
  - 操作码主要包括两部分内容：
    - 操作种类：加、减、乘、除、数据传送、移位、转移、输入输出
    - 操作数描述
      - 数据的类型：定点数、浮点数、复数、字符、字符串、逻辑数、向量
      - 进位制：2进制、10进制、16进制
      - 数据字长：字、半字、双字、字节
  - 地址码通常包括三部分内容：
    - 地址：直接地址、间接地址、立即数、寄存器编号、变址寄存器编号
    - 地址的附加信息：偏移量、块长度、跳距
    - 寻址方式：直接寻址、间接寻址、立即数寻址、变址寻址、相对寻址、寄存器寻址(可能)
- 指令格式的优化
  - 指令=操作码+地址码
  - 指令格式的优化：如何用最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最短。
  - 主要目标：
    - 节省程序的存储空间
    - 指令格式尽量规整，便于译码
- 操作码的优化表示：
  - 操作码的三种编码方法：
    - 固定长度：规整性好，解码简单，空间大
    - Huffman编码：空间小，规整性不好，解码复杂。
    - 扩展编码：折衷方案，由固定长操作码与Huffman编码法相结合形成
  - 改进操作码编码方式能够节省程序存储空间
- 指令字格式的优化：
  - 只有操作码的优化，没有在地址码和寻址方式上采取措施，程序的总位数还是难以减少。
  - 如果主存按位编址，则部分指令的读取需要两个周期，是机器速度明显下降。