第1章 计算机系统设计基础

世界上第一台通用电子数字计算机 ENIAC(Electronic Numerical Integrator And Computer),于 1946 年在美国宾夕法尼亚大学建成。由于 ENIAC 输入和更换程序特别繁杂,ENIAC 课题组的顾问、著名数学家 Von Neumann 提出将程序的指令与指令所操作的数据一起存于存储器的概念。这个著名的存储式程序(Stored-Program)概念,成为计算机工作的基本原理。这一概念也被英国的图灵大约同时期提出。世界上第一台存储式程序计算机是 1949 年在美国剑桥大学建成的 EDSAC(Electronic Delay Storage Automatic Calculator)计算机,它使用3 000 只电子管,每秒钟能完成700 次加法运算。1953 年 IBM 公司制造出第一台电子存储式程序的商用计算机。

50 多年来,计算机系统性能得到了大幅度提高,价格却大幅度下降。计算机的发展已经历了四次更新换代,现在正处于第五代。依据半导体技术发展水平,这五代划分是:

第一代:1945 ~ 1954 年,电子管和继电器;

第二代:1955 ~ 1964 年,晶体管和磁芯存储器;

第三代:1965 ~ 1974 年,中小规模集成电路(MSI-SSl);

第四代:1975 ~ 1990 年,LSI/VLSI 和半导体存储器;

第五代:1990 年至今,ULSI/GSI(Giga-Scale Integration)巨大规模集成电路。

计算机系统更新换代的标志主要是两个方面:一是由于器件不断发展,使计算机系统的工作速度、功能、集成度和可靠性等指标不断提高和价格不断降低而造成的;二是得益于计算机系统结构的改进。有人统计在 1965 ~ 1975 年期间,计算机系统性能提高近 100 倍,其中由于器件性能提高使其性能增加 10 倍,而另外的 10 倍,主要归功于系统结构改进。在 50 多年的发展进程中,器件在技术上的改进是比较稳定的,而系统结构的改进则有较大起伏,特别是近 20 年来,由于计算机系统的设计对集成电路技术的依赖性大为增加,从而使得在这一期间内,各类不同的计算机系统的性能增长率有了差异。

巨型机的性能增长得益于器件技术和系统结构两方面的改进;大型机的性能增长则主要靠器件工艺上的改进,因为系统结构方面的改进没有新的突破;小型机的发展,一方面是由于计算机实现方法有了较大的改进,另一方面是因为采用了许多大型机中行之有效的先进技术。然而这三类计算机,在 1970 ~ 1990 年期间,每年计算机的性能平均增长率均只在 18% 左右。与之形成明显对照的是微型计算机的性能增

长非常快,每年的平均增长率约为35%,这是因为微型计算机能从集成电路技术的进展中得到最为直接的好处。自20世纪80年代起,微处理机技术实际上已成为新系统结构和老系统结构更新时所选用的主要技术。

自1985年开始,一种具有新颖设计风格的系统结构,即RISC技术的系统结构,为计算机工业界所青睐。它将集成电路技术进展、编译技术改进和新的系统结构设计思想三者有机地结合起来,从而使得这种风格设计的计算机系统的性能以每年增长一倍的高速率加以改进。应该指出的是,这种改进的基础是通过对以往计算机如何被使用的模拟实验数据进行定量分析后获得的。有的学者将这种设计风格称为定量分析的计算机系统结构设计风格,显然这比传统的定性设计风格要精确得多,开发RISC技术的两位先驱者,美国加州大学伯克莱分校的D. Patterson教授和斯坦福大学的J. Hennessy教授是这种定量分析设计方法的主要倡导者。

微处理技术的发展更能说明计算机系统结构对计算机性能提高的贡献。据有关资料显示,1995年,微处理器中系统结构技术的改进,是单纯依靠电子技术进步来提高微处理器性能的5倍。而未来微处理器性能的提高,主要是依靠在微处理器中实现指令级、线程级的并行性。

1.1 计算机系统的基本概念

1.1.1 计算机系统的层次结构

现代通用计算机系统是由硬件和软件组成的一个复杂系统,按其功能可划分为多级层次结构,如图1.1所示。层次结构由上往下依次为应用语言机器级、高级语言机器级、汇编语言机器级、操作系统机器级、传统机器级和微程序机器级。对于一个具体的计算机系统,层次的多少会有所不同。

各机器级的实现主要靠翻译或解释,或者是这两者的结合。翻译(Translation)是先用转换程序将上一级机器级上的程序整个地变换成下一级机器级上可运行的等效程序,然后再在下一级机器级上去实现的技术。解释(Interpretation)则是在下一级机器级上用它的一串语句或指令来仿真上一级机器级上的一条语句或指令的功能,通过对上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。

应用语言虚拟机器级是为了满足信息管理、人工智能、图像处理、辅助设计等专门的应用来设计的。使用面向某种应用环境的应用语言(L5)编写的程序一般是经应用程序包翻译成高级语言(L4)程序后,再逐级向下实现的。高级语言机器级上的程序可以先用编译程序整个地翻译成汇编语言(L3)程序或机器语言(L1)程序,再逐级或越级向下实现,也可以用汇编语言(L3)程序、机器语言(L1)程序,甚至微指令语言(L0)程序解释实现。对汇编语言(L3)源程序则先用汇编程序整个将其变换成等效的二进制机器语言(L1)目标程序,再在传统机器级上实现。操作系统程序虽然已

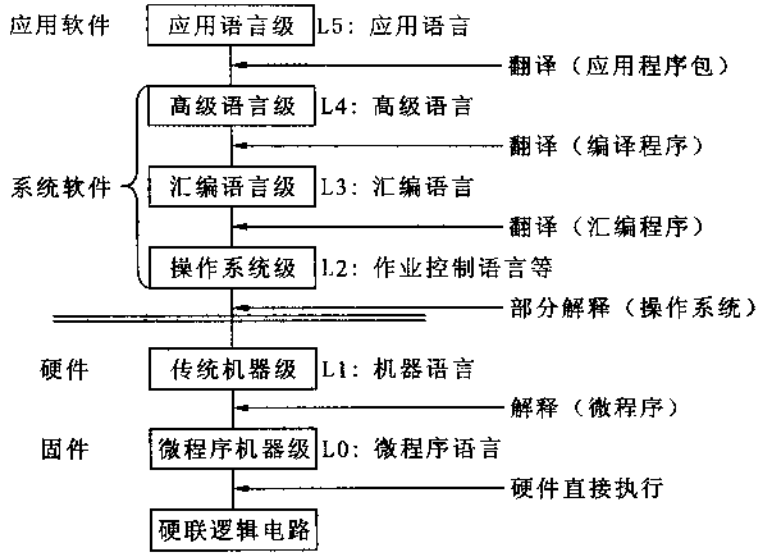


图1.1 计算机系统的多级层次结构

发展成用高级语言(如面向编写操作系统软件的C语言)编写,但最终还是要用机器语言程序或微指令程序来解释。它提供了传统机器级所没有,但为汇编语言和高级语言使用和实现所用的基本操作、命令和数据结构。例如,文件管理、存储管理、进程管理、多道程序共行、多重处理、作业控制等所用到的操作命令、语句和数据结构等。因此,操作系统机器级放在传统机器级和汇编语言机器级之间是适宜的。传统机器级采用组合逻辑电路控制,其指令可直接用硬件来实现,也可以采用微程序控制,用微指令(L0)程序来解释实现,微指令直接控制硬件电路的动作。

就目前的状况来看,第0级用硬件实现,第1级用微程序(固件)实现,第2级到第5级大多用软件实现。我们称以软件为主实现的机器为虚拟机器,以区别于由硬件或固件实现的实际机器。虚拟机器不一定全都由软件实现,有些操作可以用固件或硬件实现。如操作系统中的某些命令可由比它低两级的微程序解释,或全部用硬件来实现。高级语言机器则直接用微程序解释或用硬件实现,没有编译软件。

计算机系统结构作为一门学科,主要研究软件、硬件功能分配和对软件、硬件界面的确定,即哪些功能由软件完成,哪些功能由硬件完成。采用何种实现方式,要从整个计算机系统的效率、速度、价格、资源状况等方面考虑,对软件、硬件、固件取舍进行综合平衡。软件和硬件在逻辑功能上是等效的。原理上,软件实现的功能完全可以用硬件或固件完成,硬件实现的功能也可以由软件的模拟来完成,只是其性能、价格、实现的难易程度等有所不同。具有相同功能的计算机系统,其软、硬件功能分配比例可以在很宽的范围内变化。

1.1.2 计算机系统结构

“计算机系统结构”这个名词来源于英文 Computer Architecture，也有译成“计算机体系结构”的。Architecture 这个字原来用于建筑领域，其意义是“建筑学”、“建筑物的设计或式样”，它是指一个系统的外貌。20 世纪 60 年代这个名词被引入计算机领域，“计算机系统结构”一词已经得到普遍应用，它研究的内容不但涉及计算机硬件，也涉及计算机软件，已成为一门学科。但对“计算机系统结构”一词的含义仍有多种说法，并无统一的定义。

计算机系统结构这个词是 G. M. Amdahl 等人在 1964 年提出的。他们把系统结构定义为由程序设计者所看到的一个计算机系统的属性，即概念性结构和功能特性。这实际上是计算机系统的外特性。按照计算机层次结构，不同程序设计者所看到的计算机有不同的属性。例如，对于使用 FORTRAN 高级语言的程序员来讲，一台 IBM3090 大型机、一台 VAX-11/780 小型机或一台 PC 微型机，看起来都是一样的，因为在这三台计算机上运行他所编制的程序，所得到的结果是一样的。但对于使用汇编语言程序的程序员来讲，由于这三台机器的汇编语言指令完全不一样，他所面对的计算机的属性就也会不一样。另外，即使对同一台机器来讲，处在不同级别的程序员，例如应用程序员、高级语言程序员、系统程序员和汇编程序员，他们所看到的计算机外特性也是完全不一样的。那么通常所讲的计算机系统结构的外特性应是处在哪一级的程序员所看到的外特性呢？比较一致的看法是机器语言程序员或编译程序编写者所看到的外特性，这种外特性是指由他们所看到的计算机基本属性，即计算机的概念性结构和功能特性，这是机器语言程序员或编译程序编写者为使其所编写、设计或生成的程序能在机器上正确运行所必须遵循的。由机器语言程序员或编译程序编写者所看到的计算机的基本属性是指传统机器级的系统结构，在传统机器级之上的功能被视为属于软件功能，而在其之下的则属于硬件和固件功能。因此，计算机系统的概念性结构和功能属性实际上是计算机系统中软、硬件之间的界面。

在计算机技术中，一种本来是存在的事物或属性，但从某种角度看似乎不存在，称为透明性现象。通常，在一个计算机系统中，低层机器级的概念性结构和功能特性，对高级语言程序员来说是透明的。由此看出，在层次结构的各个级上都有它的系统结构。

就目前的通用机来说，计算机系统结构的属性应包括以下几个方面：

- 硬件能直接识别和处理的数据类型和格式等的数据表示；
- 最小可寻址单位、寻址种类、地址计算等的寻址方式；
- 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；
- 二进制或汇编级指令的操作类型、格式、排序方式、控制机构等的指令系统；
- 内存的最小编址单位、编址方式、容量、最大可编址空间等的存储系统组织；
- 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；

- 系统机器级的管态和用户态的定义和切换；
- 输出设备的连接、使用方式、流量、操作结束、出错指示等的机器级 I/O 结构；
- 各部分的信息保护方式和保护机构，等等。

1.1.3 计算机组成与实现

计算机组成(Computer Organization)指的是计算机系统结构的逻辑实现，也常称为计算机组织。它包括机器级内的数据流和控制流的组成以及逻辑设计等。它着眼于机器级内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。计算机组成设计要解决的问题是在所希望达到的性能和价格下，怎样最佳、最合理地把各种设备和部件组织成计算机，以实现所确定的系统结构。计算机组成设计主要是围绕提高速度，着重从提高操作的并行度、重叠度以及分散功能和设置专用功能部件来进行的。

计算机组成设计要确定的方面一般应包括：

- 数据通路宽度(在数据总线上一次并行传送的信息位数多少)；
- 专用部件的设置(设置哪些专用部件，如乘除法专用部件、浮点运算部件、字符处理部件、地址运算部件等，每种专用部件设置的数量，这些都与机器所需达到的速度、专用部件的使用频度高低及允许的价格等有关)；
- 各种操作对部件的共享程度(共享程度高，即使操作在逻辑上不相关也只能分时使用，限制了速度，但价格便宜，可以设置多个部件降低共享程度，提高操作并行度来提高速度，但价格也将提高)；
- 功能部件的并行度(功能部件的控制和处理方式是采用顺序串行，还是采用重叠、流水或分布处理)；
- 控制机构的组成方式(事件、操作的排序机构是采用硬件控制还是用微程序控制，是采用单机处理还是用多机处理或功能分布处理)；
- 缓冲和排队技术(在不同部件之间怎样设置及设置多大容量的缓冲器来弥补它们的速度差异，是采用随机方式，还是先进先出、先进后出、优先级或循环方式来安排等待处理事件的先后顺序)；
- 预估、预判技术(为优化性能和优化处理，采用什么原则来预测未来的行为)；
- 可靠性技术(采用什么样的冗余技术和容错技术来提高可靠性)，等等。

计算机实现(Computer Implementation)指的是计算机组成的物理实现。它包括处理机、主存储器等部件的物理结构，器件的集成度、速度和信号，器件、模块、插件、底板的划分与连接，专用器件的设计，电源、冷却、装配等技术。

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令系统及其执行模型；计算机组成是计算机系统结构的逻辑实现；计算机实现是计算机组成的物理实现。它们各自包含不同的内容和采用不同的技术，但又有紧密的联系。在学习和理解时有两点需要注意：

一是计算机系统结构、组成和实现之间的界限变得越来越模糊了。尤其是严格区分计算机系统结构和组成已不太可能，也没有太大的实际意义。随着 VLSI 技术的进步，新器件的不断涌现，当今计算机系统结构的设计所面临的问题与 G. M. Amdahl 所处的时期大不相同，就是与十年前也大不相同。例如，十年前系统配置几十至几百 KB 的内存就很不错了，某些指令系统的设计中甚至有对存储器操作数直接进行加减的指令，不惜牺牲执行速度来珍惜宝贵、有限的内存资源。现在，存储器芯片的集成度大幅度提高而价格急剧下降，内存容量已不是计算机系统结构设计的主要问题了；如何组织存储器以提高存取速度，如何保证 CPU 与内存之间的通道不成为系统性能的瓶颈，是当代计算机系统结构设计必须考虑的问题。现在，一般已将功能模块设计移入计算机系统结构考察的范畴之内。

二是我们介绍了计算机系统结构、组成和实现三者之间的关系，但不要认为计算机系统结构设计就是硬件设计，两者不能混淆。操作系统、编译程序以及高级语言的发展都对计算机系统结构的设计有重要影响。计算机系统结构设计是在功能这一层次上考虑问题，当然要涉及硬件，但它不是只包括硬件设计。例如，存储器管理功能可以由硬件和软件共同实现，它们之间的分工取决于当前硬件和软件的可用性、性能和价格。在 VLSI 发展的初期，存储器管理功能一般由软件实现；现在，存储器控制芯片已能实现这些存储器管理算法并维护存储器与高速缓存的一致性。因此，计算机系统结构设计的一个主要任务是研究软件、硬件功能分配和对软件、硬件界面的确定。

1.1.4 计算机系统结构的分类

研究计算机系统的分类方法有助于认识计算机系统结构的组成和特点，理解计算机系统的工作原理和性能。常用的计算机系统结构分类方法有以下三种：

1. Flynn 分类法

1966 年 M. J. Flynn 提出了按照指令流和数据流的多倍性概念进行分类的方法，其定义如下：

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，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可并行执行的指令或数据的最大可能个数。

按照指令流和数据流的不同组织方式，把计算机系统的结构分为以下四类：

(1) 单指令流单数据流 SISD(Single Instruction stream Single Data stream)。传统的计算机就属于这一类。

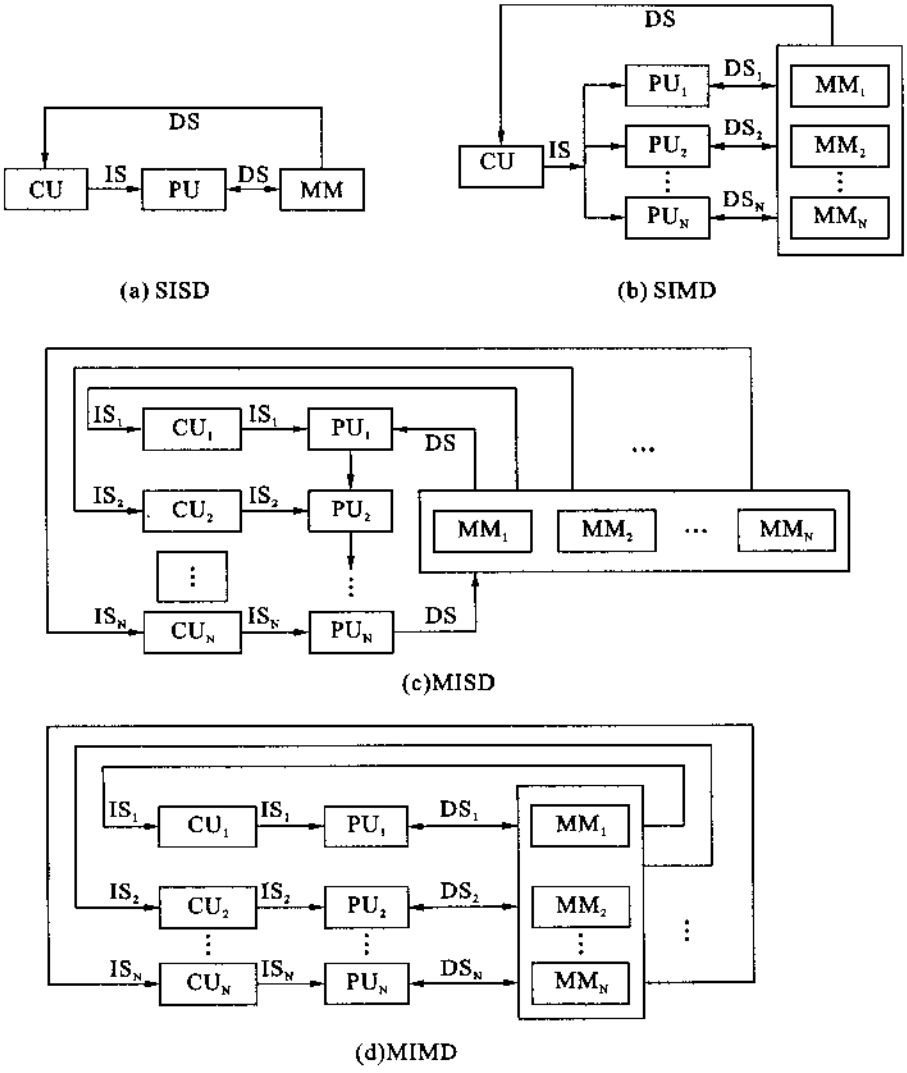
(2) 单指令流多数据流 SIMD(Single Instruction stream Multiple Data stream)。ILLIAC-Ⅳ阵列计算机是典型的这类机器。

(3) 多指令流单数据流 MISD(Multiple Instruction stream Single Data stream)。对

于这类计算机的分类是有争议的，有的说这类机器根本不存在，有的将高级流水技术实现的计算机（如 VLIW）划分为这一类。

（4）多指令流多数据流 MIMD（Multiple Instruction stream Multiple Data stream）。多计算机系属于这一类。

Flynn 分类法各机器结构如图 1.2 所示。



CU：控制部件　PU：处理部件　MM：存储模块　IS：指令流　DS：数据流

图 1.2　Flynn 分类法各机器结构

2. 冯氏分类法

1972 年美籍华人冯泽云(Tseyun Feng)教授提出了用最大并行度(Pm)来定量描述各种计算机系统特性的冯氏分类法。最大并行度 Pm 的定义为:计算机系统在单位时间内能够处理的最大的二进制位数。

图 1.3 描述了用最大并行度对计算机系统分类的方法。用平面直角坐标系中的一个点代表一个计算机系统,横坐标代表字宽(n 位),即在一个字中同时处理的二进制位数;纵坐标代表位片宽度(m 位),即在一个位片中能同时处理的字数。

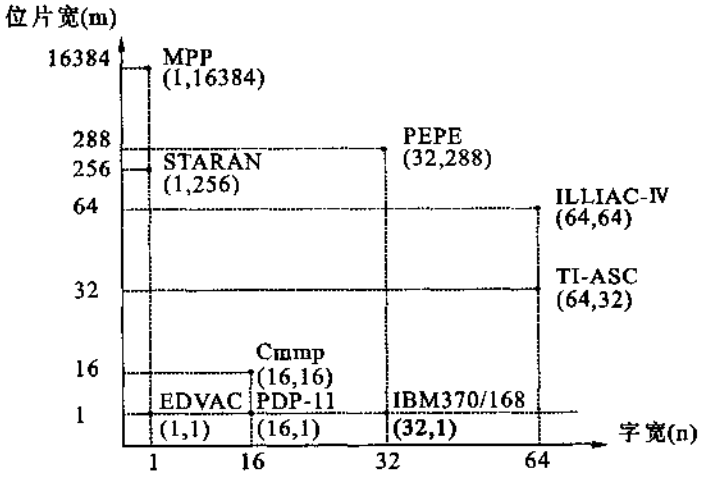


图 1.3 按最大并行度的冯氏分类法

由图 1.3 可得出如下四类不同处理方法的计算机系统结构:

(1) 字串位串 WSBS(Word Serial and Bit Serial),其中 $n=1, m=1$。这是第一代计算机发展初期的纯串行计算机。

(2) 字并位串 WPBS(Word Parallel and Bit Serial),其中 $n>1, m=1$。这是传统并行单处理机。

(3) 字串位并 WSBP(Word Serial and Bit Parallel),其中 $n=1, m>1$。STARAN, MPP, DAP 属于这种结构。

(4) 字并位并 WPBP(Word Parallel and Bit Parallel),其中 $n>1, m>1$。PEPE, ILLIAC-Ⅳ, Cmmp 属于这种结构。

3. Handler 分类法

1977 年 Wolfgan Handler 根据并行度和流水线提出了另一种分类法。这种分类方法把计算机的硬件结构分成三个层次,并分别考虑它们的可并行-流水处理程度。这三个层次是:

(1) 程序控制部件(PCU)的个数 k;

(2) 算术逻辑部件(ALU)或处理部件(PE)的个数 d;

(3) 每个算术逻辑部件包含基本逻辑线路(ELC)的套数 w。

这样我们可以把一个计算机系统的结构用如下公式表示:t(系统型号) = (k,d,w)。为了进一步揭示流水线的特殊性,一个计算机系统的结构可用如下公式表示:t(系统型号) = (k×k',d×d',w×w'),其中:k'表示宏流水线中程序控制部件的个数,d'表示指令流水线中算术逻辑部件的个数,w'表示操作流水线中基本逻辑线路的套数。

例如,CRAY-1 有 1 个 CPU,12 个相当于 ALU 或 PE 的处理部件,可以最多实现 8 级流水线。字长为 64 位,可以实现 1~14 位流水线处理。所以 CRAY-1 的系统结构可表示为:

t(CRAY-1) = (1,12×8,64(1~14))

下面是用这种分类法的例子:

t(PDP-11) = (1,1,16)

t(ILLIAC-Ⅳ) = (1,64,64)

t(STARAN) = (1,8192,1)

t(Cmmp) = (16,1,16)

t(PEPE) = (1×3,288,32)

t(TIASC) = (1,4,64×8)

1.2 计算机系统设计技术

1.2.1 计算机系统设计原理

下面介绍计算机系统设计中经常用到的几个定量原理。

1. 加快经常性事件的速度(Make the Common Case Fast)

这是计算机设计中最重要也最广泛采用的设计准则。使经常性事件的处理速度加快能明显提高整个系统的性能。一般说来,经常性事件的处理比较简单,因此比不经常出现的事件处理起来要快。例如,在 CPU 中两个数进行相加运算时,相加结果可能出现溢出现象,也可能无溢出发生,显然经常出现的事件是不发生溢出的情况,而溢出是偶然发生的事件。因此,在设计时应优化不发生溢出的情况,使这个经常性事件的处理速度尽可能快,而对溢出处理则不必过多考虑优化。因为发生溢出的概率很小,即使发生了,处理得慢一些也不会对系统性能产生很大的影响。

在计算机设计中经常会遇到上述情况。那么,如何确定经常性事件以及如何加快处理它,这就是下面介绍的由 G. M. Amdahl 提出的定律要解决的问题。

2. Amdahl 定律

Amdahl 定律:系统中对某一部件采用某种更快的执行方式后整个系统性能的改

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。

$$\text{系统加速比}=\frac{\text{系统改进后的性能}}{\text{系统改进前的性能}}=\frac{\text{系统改进前执行某一任务的总时间}}{\text{系统改进后执行同一任务的总时间}}$$

Amdahl 定律定义了采取改进（加速）某部分功能后可获得的性能改进或执行时间的加速比。系统加速比取决于以下两个因素：

（1）可改进部分在原系统计算时间中所占的比例。例如，一个需运行 60s 的程序中有 20s 的运算可以加速，那么该比例就是 20/60。这个值用“可改进比例”表示，它总是小于或等于 1 的。

（2）可改进部分改进以后的性能提高。例如，系统改进后执行程序，其中可改进部分花费的时间为 2s，而改进前该部分需花费的时间为 5s，则性能提高为 5/2。用“部件加速比”表示性能提高比，一般情况下它是大于 1 的。

若以 T_e 和 T_o 分别表示采取某种改进措施前后完成同一任务所需的总时间，用 f_e 表示可改进部分的比例（$0 \leqslant f_e \leqslant 1$），用 r_e 表示采用改进措施比不采用改进措施可加快执行的倍数，则上述各参量与系统加速比（S_p）之间的关系可用如下表达式表示：

$$T_o = T_e \times (1-f_e) + \frac{f_e \times T_e}{r_e} = \left[(1-f_e) + \frac{f_e}{r_e}\right] \times T_e$$

（部件改进后，系统的总执行时间等于不可改进部分的执行时间加上可改进部分改进后的执行时间）

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

例 1.1 若考虑将系统中的某一功能的处理速度加快到 10 倍，该功能部件的原处理时间为整个系统运行时间的 40%，则采用改进措施后，使整个系统的性能提高了多少？

解：由题意可知：$f_e = 0.4$，$r_e = 10$，由加速比公式可得

$$S_p = \frac{1}{(1-0.4) + 0.4/10} = 1.56$$

3. 程序访问的局部性规律

所谓程序访问的局部性是指程序执行中，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。统计表明一个程序执行时间中的 90% 是花费在 10% 程序代码上。例如对 Gcc（GnuC 编译）、Spice（CAD 电路分析软件）以及 TeX（文本处理软件）三个典型测试程序的测试表明，相应的比例数分别为 13%，9.5% 和 9.3%，遵从上述的 90%/10% 定量规律。

程序访问局部性主要反映在时间和空间局部性两个方面，时间局部性是指程序中近期被访的信息项很可能马上将被再次访问；空间局部性是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。

程序访问局部性规律是按层次构成存储体系的主要依据，程序中的大部分可放在容量较大，工作速度较慢及成本较低的慢速存储部件中，而只需将其中的一小部分（如10%～20%）存放在高速的存储部件中，这便是虚存、Cache高速缓存得以实现的具体根据。

1.2.2 计算机系统设计方法

1. 软硬件舍取的基本原则

计算机系统结构设计的一个主要任务是进行软、硬件功能分配。一般来说，提高硬件功能的比例可提高解题速度，减少程序所需存储空间，但会提高硬件成本，降低硬件利用率和计算机系统的灵活性和适应性；而提高软件功能比例可降低硬件成本，提高系统的灵活性、适应性，但解题速度下降，软件设计费用和所需存储空间增加。下面介绍软、硬件舍取的三个基本原则。

（1）在现有硬件和器件（主要是逻辑器件和存储器件）条件下，系统要有高的性价比，主要从实现费用、速度和其他性能要求来综合平衡；

（2）系统结构和组成的设计尽可能地不要过多或不合理地限制各种组成、实现技术的采用；

（3）不仅从“硬”的角度考虑如何应用组成技术的成果和便于发挥器件技术的进展，还应从“软”的角度把如何为编译和操作系统的实现以及为高级语言程序的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

计算机系统的设计，不仅应该关心系统结构所包含的指令系统设计，还应考虑功能的组成、逻辑设计以及具体的物理实现，它包括了集成电路的设计、封装、电源以及冷却等。

计算机系统设计者的主要任务如下：

（1）要满足用户对功能上的要求以及相应的对价格和性能的要求。这里功能上的要求包括：应用领域，如专用还是通用，科学计算或是商业上使用；软件兼容级别，在程序设计语言级或是目标代码（二进制）级兼容；操作系统需求，如地址空间的大小、存储器管理、保护及环境转换；中断和自陷以及对标准的要求，如浮点标准、I/O总线标准、操作系统标准、网络标准及程序设计语言标准等。

（2）在满足功能要求基础上，进行设计的优化。优化主要是以性能价格比为衡量指标，仔细考虑某一所需的功能应该用软件还是用硬件来实现。因此在系统设计中如何很好地平衡软/硬件二者就特别关键了。此外在选择硬件或软件的方法时，还应考虑设计复杂性和具体实现的难易程度。

（3）设计应能适应日后发展趋势。这是指好的系统结构设计应能经受硬件技术、软件技术以及应用需求特征的变化，因此设计者应对计算机使用趋势和计算机技术的发展趋势有足够的认识。

硬件技术进展表现在IC(集成电路)逻辑技术、DRAM技术及Disk(硬盘)技术。统计资料表明:一个芯片上的晶体管数大约每年增加25%,因此每三年可增加1倍;器件的开关速度增长基本类似DRAM的密度每年增长约60%,因此每三年将增长3倍;访问存储器周期改进相应较慢,每十年约减少1/3;硬盘密度每年增加25%,每三年增加1倍;访问时间则每十年减少1/3。

软件技术的发展趋向:一是程序所要求的存储器空间增长,大约每年增长1.5~2倍,因此相应地要求地址位每年能增长0.5位到1位。这两点对设计非常重要,即必须留有足够的地址空间以待扩展。二是汇编语言将逐步为高级语言所替代,编译技术将起更大作用,因此系统结构应能更好地支持编译要求。编译程序将逐步成为用户和计算机间的主要界面。

3. 计算机系统设计的基本方法

从计算机系统的多级层次结构出发,可以有"由上往下"、"由下往上"和"由中间开始"三种不同的设计思路。

(1)"由上往下"设计是先考虑如何满足应用要求,确定好面对使用者那级机器应有什么基本功能和特性,如基本命令、指令或语言结构、数据类型和格式等,然后再逐级往下设计,每级都考虑怎样优化上一级实现。这样设计的计算机系统对于设计时所面向的应用必然是很好的。因此,它适合于专用机设计,但不宜用于通用机的设计,因为当应用对范围改变时,软、硬件分配会很不适应,而使系统效率急剧下降。

(2)"由下往上"设计是不管应用要求,只根据能拿到的器件,参照或吸收已有各种机器的特点,先设计出微程序机器级(如果采用微程序控制)及传统机器级,然后再为不同应用配多种操作系统和编译系统软件,使应用人员可根据所提供的语言种类、数据形式,采用合适的算法来满足相应的应用。这是20世纪60~70年代前常用的通用机设计思路。

由于是在硬件不能改变的情况下去被动设计软件,尽管某些硬件的设计不利于软件的实现,而且有时只需稍许改变或增加某些功能就可大大简化软件的设计,也无法加以改变。因此,由下往上设计在硬件技术飞速发展而软件发展相对缓慢的今天,难以适应系统设计要求,所以已很少使用。

(3)"由中间开始"设计的"中间"指的是层次结构中的软、硬件交界面,目前多数是在传统机器级与操作系统机器级之间。

进行合理的软、硬件功能分配时,既要考虑能拿到的硬件及器件,又要考虑可能的应用所需的算法和数据结构,先定义好这个交界面。确定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、编译系统的实现提供些什么支持。然后由这个中间点分别往上、往下进行软件和硬件的设计。软件人员依次设计操作系统级、汇编语言级、高级语言级和应用语言级;硬件人员依次设计传统机器级、微程序机器级、数字逻辑级。

软件和硬件并行设计,可缩短系统设计周期,设计过程中可交流协调,是一种交

互式的、较好的设计方法。当然,这要求设计者应同时具备丰富的软、硬、器件和应用等方面的知识。又由于软件设计周期一般比较长,为了能在硬件研制出来之前开展软件的设计测试,还应有有效的软件设计环境和开发工具,如在某个宿主机上建立目标机的指令模拟器、系统结构分析模拟器及良好的调试程序、性能评价的测试程序等。

1.3 计算机系统的性能评价

衡量计算机系统性能可有各种指标,但最为关键的是时间。时间可根据计算方法给以不同的定义,如响应时间、CPU 时间等。响应时间是指在用户向计算机系统送入一个任务后,直到获得他所需要的结果所需的等待时间,其中包括了访问磁盘和访问主存储器时间、CPU 运算时间、I/O 动作时间以及操作系统工作的时间开销等。虽然这种定义比较直观,但对于多道程序,由于 CPU 可在某一程序等待 I/O 操作时转去执行其他程序,响应时间并不能区别这种情况。另一种情况是只考虑 CPU 时间,此时便可加以区别,它将不包括等待 I/O 操作的时间以及 CPU 转去运行其他程序所用的时间。当然 CPU 时间本身还可分为用户 CPU 时间和系统 CPU 时间。系统 CPU 时间的统计很难做到精确,因为这实际上是要求操作系统进行自测量。此外,当比较具有不同系统代码的机器时,由于系统 CPU 时间是不一样的,因面误差较大,故采用用户 CPU 时间作为性能衡量时间较为妥当。当然,在衡量未加载系统的性能时,采用前述的响应时间较为合适,而衡量 CPU 性能则宜采用用户 CPU 时间。下面主要讨论以用户 CPU 时间来衡量的 CPU 性能。

1.3.1 CPU 性能

绝大多数计算机都使用以固定速率运行的时钟,它的运行周期称为时钟周期(Clock Cycles),它常以时间长短(以 ns 计算)或运行速率(以 MHz 计算)来表示。一个程序在 CPU 上运行所需的时间 T_{CPU},可用以下的公式表示:

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

式中,I_N 表示要执行程序中的指令总数,CPI(Cycles Per Instruction)表示执行每条指令所需的平均时钟周期数,而 T_C 表示时钟周期的时间。

由此公式可见,用户 CPU 时间取决于三个方面:时钟周期(或速率),每条指令所需时钟周期数以及程序中总的指令数。其中,I_N 主要取决于机器指令系统和编译技术,CPI 主要与计算机组成和指令系统有关,而 T_C 则主要由硬件工艺和计算机组成决定。

每条指令平均所需时钟周期数 CPI,可由下式表示:

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

式中,I_i 表示第 i 类指令在程序中执行次数,CPI_i 表示执行一条第 i 类指令所需的平

均时钟周期数，n 为程序中所有的指令种类数，I_i/I_N 表示第 i 种指令在程序中所占的比例。

下面通过两个例子来说明上述概念和公式的应用。

例 1.2 假定在设计机器的指令系统时，对条件转移指令的设计有以下两种不同的选择：

CPU_A 采用一条比较指令来设置相应的条件码，由紧随其后的转移指令对此条件码进行测试，以确定是否要进行转移。显然为实现一次条件转移就必须使用比较和测试两条指令。

CPU_B 采用使转移指令不仅具有判别是否实现转移功能，还包含有上述方案中的比较指令的功能，这样实现一次条件转移只需要一条指令就可完成。

若假设在两个机器的指令系统中，执行条件转移指令需 2 个时钟周期，而其他的指令只需 1 个时钟周期。又假设在 CPU_A 上，要执行的指令中有 20% 是条件转移指令，由于每条转移指令都需要一条比较指令，因此，比较指令也将占据 20%。由于 CPU_B 在转移指令中包含了比较功能，因此它的时钟周期就比 CPU_A 的要慢 25%。现在要问，采用不同转移指令方案的 CPU_A 和 CPU_B，哪一个工作速度会更快些？

解：根据上述假设，可计算得 $CPI_A = 0.2 \times 2 + 0.8 \times 1 = 1.2$，因为转移指令需 2 个时钟周期，而其余的指令，包括比较指令，只需 1 个时钟周期，所以

$$T_{CPU_A} = I_{NA} \times 1.2 \times T_{CA} = 1.2 I_{NA} \times T_{CA}$$

对于 CPU_B，由于没有比较指令，从而使转移指令由原来的占 20% 上升为 20% ÷ 80% = 25%，它也需 2 个时钟周期，而其余的 75% 指令只需 1 个时钟周期，因而有 $CPI_B = 0.25 \times 2 + 0.75 \times 1 = 1.25$。由于 CPU_B 中没有比较指令，因此 $I_{NB} = 0.8 \times I_{NA}$。此外，$T_{CB} = 1.25 T_{CA}$，所以

$$T_{CPU_B} = I_{NB} \times CPI_B \times T_{CB} = 0.8 I_{NA} \times 1.25 \times 1.25\ T_{CA} = 1.25\ I_{NA} \times T_{CA}$$

与 T_{CPU_A} 相比较，可见由于 CPU_A 所需时间较少，所以 CPU_A 比 CPU_B 运行得更快些。

例 1.3 在上例中，如果 CPU_B 的时钟周期只比 CPU_A 的慢 10%，那么此时，哪一个 CPU 会工作得更快些？

解：$T_{CPU_A} = 1.2\ I_{NA} \times T_{CA}$，而此时因 $T_{CB} = 1.10 T_{CA}$，故

$$T_{CPU_B} = 0.8 I_{NA} \times 1.25 \times 1.10\ T_{CA} = 1.10\ I_{NA} \times T_{CA}$$

由于 CPU_B 所需时间较少，故 CPU_B 比 CPU_A 运行得更快些。

1.3.2 MIPS 和 MFLOPS

除了时间评估标准之外，MIPS 和 MFLOPS 也是比较常用的性能评估标准。

1. MIPS

MIPS(Million Instructions Per Second，每秒百万次指令)是一个用来描述计算机性能的尺度。对于给定的一个程序，MIPS 可表示成：

$$\text{MIPS} = \frac{I_N}{T_E \times 10^6} = \frac{I_N}{I_N \times \text{CPI} \times T_C \times 10^6} = \frac{R_C}{\text{CPI} \times 10^6}$$

式中，T_E 表示执行该程序所需时间；R_C 表示时钟速率，它是时钟周期时间 T_C 的倒数。

在使用 MIPS 时应注意它的应用范围，它只适宜于评估标量机，因为在标量机中执行一条指令，一般可得到一个运算结果，而向量机中，执行一条向量指令通常可得到好多个运算结果，因此，用 MIPS 来衡量向量机是不合适的。在 MIPS 中，不仅是运算指令，所有的服务性指令，如取数、存数、转移等都计算在内，而在浮点运算中服务性指令均不予计入。

有时还用相对 MIPS_{ref} 这一标准，这需要事先选择一个给定的参照计算机的性能，然后与其比较。因此有：

$$\text{MIPS}_{\text{ref}} = \frac{T_{\text{ref}}}{T_V} \times \text{MIPS}_{\text{ref}}$$

式中，T_{ref} 表示在参照机上程序的执行时间；T_V 表示相同程序在要评估机器上执行时间；MIPS_{ref} 表示所约定的参照机的 MIPS 速率。在 20 世纪 80 年代，常以 DEC 公司的 VAX-11/780 计算机作为参照机，称其为 1 MIPS 机器。

2. MFLOPS

MFLOPS(Million FLoating point Operations Per Second，每秒百万次浮点运算)可用如下式子表示：

$$\text{MFLOPS} = \frac{I_{FN}}{T_E \times 10^6}$$

式中，I_{FN} 表示程序中的浮点运算次数。

MFLOPS 测量单位比较适用于衡量向量机的性能，因为一般而言，同一程序运行在不同计算机上时往往会执行不同数量的指令数，但所执行的浮点数个数常常是相同的。采用 MFLOPS 作为衡量单位时，应注意它的值不但会随整数、浮点数操作混合比例的不同发生变化，而且也会随快速和慢速浮点操作混合比例的变化而变化。例如，运行由 100% 浮点加法组成的程序所得到的 MFLOPS 值将比由 100% 浮点除法组成的程序要高。

MFLOPS 和 MIPS 两个衡量值之间的量值关系没有统一的标准，一般认为在标量计算机中执行一次浮点运算需 2 ~ 5 条指令，平均约需 3 条指令，故有 1MFLOPS ≈ 3MIPS。

1.3.3 基准测试程序

计算机系统的评价除了与被评价的机器的结构、功能等特性参数有关以外，还与输入，即该计算机系统的工作负荷(Workload)有密切关系。被评价的一个计算机系统往往对某一种工作负荷表现出较高性能，而对另一种工作负荷则可能呈现较低性

能。为了对计算机系统的性能进行客观的评价，就需要选取具有真实代表性的工作负荷。通常采用不同层次的基准测试程序(Benchmark)来评价系统性能。

(1) 采用实际应用程序。例如，C 语言的各种编译程序 Gcc、Tex 正文处理软件以及 Spice 那样的 CAD 工具软件。

(2) 采用核心程序。这是从实际程序中抽取少量关键循环程序段，并以此来评估性能，例如 Livermore 24 Loops(24 个循环段)和 Linpack(解线性方程组)便是典型代表，但这些核心程序，只具有评价性能价值。

(3) 合成测试程序。它类似于核心程序方法，但这种合成测试程序是人为编制的，较流行的合成测试程序有 Whetstone 和 Dhrystone 两种。

Whetstone 是有关整、浮点运算的合成混合，还包括了超越函数、条件转移、函数调用。早先用 Algol 60 书写，后改用 FORTRAN 书写。

Dhrystone 主要是有关整数计算，包括字符串及数组处理。应该指出的是由于这类典型测试程序是人为编制的，因此它比核心程序离实际程序更远。

1988 年以来，美国 HP、DEC、MIPS 以及 SUN 公司等发起成立了 The Standard Performance Evaluation Corporation (SPEC，标准性能评估机构)，一致同意用一组实用程序和相应输入来评价计算机系统性能。SPEC 典型测试程序由以下 10 个程序组成：GCC, Espresso, Splce2g6, DODUC, NASA7, Li, Eqntott, Matrix300, FPPPP, TOMCATV，其中 4 个用 C 语言编写(GCC，Espresso，Li 和 Eqntott)，进行整数运算，余下 6 个用 FORTRAN 语言书写，进行浮点运算，计算所得 SPECmark 的分值越大越好，它是相对于 VAX-11/780 的性能，1 SPEC 分值相当于 0.2 ~ 0.3MFLOPS。SPEC 还可进一步分为 $SPEC_{int}$(整数 SPEC)和 $SPEC_{fp}$(浮点 SPEC)，以及与它相对应的 1989、1992、1995 和 2000 等多个版本。

类似于 SPEC 的组织还有 Perfect 俱乐部，它主要由大学和公司中对并行计算感兴趣的爱好者组成，选定了称为 Perfect Club 的典型测试程序。

1.3.4 性能评价结果的统计和比较

关于计算机性能的评价，通常用峰值性能(Peak Performance)及持续性能(Sustained Performance)两个指标。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，它不能真实反映系统的实际性能。持续性能又称为实际性能，它的值往往只有峰值性能的 5% ~35%(因算法而异)。

持续性能的表示常用算术平均(Arithmetic Mean)、几何平均(Geometric Mean)和调和平均(Harmonic Mean)三种平均值方法，这三种性能值(运算速率)的计算公式如下：

1. 算术性能平均值 A_m

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

(若以执行时间表示性能,则有 $A_m = \frac{1}{n_1}\sum_{i=1}^{n} T_i$)

2. 几何性能平均值 G_m

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

3. 调和性能平均值 H_m

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

以上三个公式中,R_i 表示由 n 个程序组成的工作负荷中执行第 i 个程序的速率,T_i 表示执行第 i 个程序所需的时间,这里 $R_i = 1/T_i$。

如果考虑工作负荷中各程序不会以相等比例出现这一情况,就需要对各程序的执行速率或执行时间加上相应权值,此时有:

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

$$G_m = \prod_{i=1}^{n} (R_i)^{wi} = (R_1)^{w1} \times (R_2)^{w2} \times \cdots \times (R_n)^{wn}$$

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

从上述的三种表示方式中,可以看到只有 H_m 值是真正与运行所有典型测试程序所需的时间总和成反比关系的,若以前面提及的衡量性能指标的惟一标准是时间这一点来看,用 H_m 值来衡量计算机系统性能是较为精确的。但 G_m 表示法有一个很好的特性:

$$\frac{G_m(X_i)}{G_m(Y_i)} = G_m\left(\frac{X_i}{Y_i}\right)$$

即几何平均比与比的几何平均是相等的,因此在对各种机器性能比较而进行性能规格化(即以某台机器性能作参考标准,其他机器性能除以该参考标准所得到的比值)过程中,不论取哪一台作参考机,G_m 均能保持比较结果的一致性,A_m 和 H_m 由于没有这样的特性,因而在作比较时,就不如 G_m 那样方便。下面通过一个例子来说明这一点。

表 1.1 中列出了用两个基准测试程序对三台计算机进行测试所获得的运行时间。由表中可见,对基准测试程序比 $B1$,Y 机的速度为 X 机的 2 倍,但对基准测试程序 $B2$,Y 机速度仅为 X 机的一半。类似地有,对 $B1$,Z 机速度仅为 X 机的一半;对 $B2$,Z 机的速度则为 X 机的 2 倍。直观地看,这三台机器应有相同的性能。然而,如果将这些测试值以 X 机为基准进行规格化(表中括号中的值),并求出相应的 A_m 值,就会发现:Y 机和 Z 机速度均比 X 机要慢 25%。