第一篇
总体功能设计

第一章　体系结构

这一章将分三节介绍体系结构的设计工作。第一节计算机体系结构主要阐述了它的概念与分类，强调了体系结构与实现的区别和关系，推动体系结构发展的因素以及当前VLSI体系结构发展的现状与趋势；第二节编译程序与RISC体系结构，主要从数据类型、寻址方式、寄存器摸型、存储器管理与核心指令集合等五个方面讨论了体系结构设计中的第一个问题——编译程序接口问题；第三节操作系统与RISC体系结构，从存储器管理、中断处理与处理机管理等三个方面讨论了体系结构设计中的第二个问题——操作系统接口问题，并按嵌入式32位RISC微计算机实时内核的要求，讨论了系统操作指令与中断机构的方案。

1.1　计算机体系结构

计算机体系结构一词是在60年代初期由IBM System/360系列机的设计者们首先提出来的。他们用这个词表示由程序员观察的机器属性，也就是机器的概念结构与功能行为，以区别于计算机内部组织，以及逻辑与物理设计。其实，1945年Von Neumann的EDVAC机报告中就强调了计算机的逻辑原理与组织，对计算机的功能与实现作了清楚的区分。由于计算机实现的内部结构与行为同其外部特性对计算机设计者来说是同样重要的，体系结构一词也常用来表示计算机的实现方面。为了区别，计算机的功能方面常叫做外体系结构(exo-Architecture)、宏体系结构(macro-Architecture)，或简称体系结构；计算机的实现方面常叫做内体系结构(endo-Architecture)、微体系结构 (Micro-Architecture)，或简称计算机组织实现。

外体系结构的主要研究内容是指令集合与存储器管理，以定义具体机器与软件之间的接口，供操作系统、编译程序以及汇编语言程序等的设计者使用。因此，外体系结构的研究与程序设计语言及编译技术的研究是紧密相关的。工程驱动的命令语言的开发在于使计算机上的映射容易，是以面向计算机外体系结构的方式表达思想的；说明语言的开发在于适合于表达人的思想，是一种更高级的语言。由于高级语言与目标机指令集合之间存在着一个软件间隙，必须由编译程序来解决，因此，计算机外体系结构设计与相应编译程序设计之间必然存在明显的相互作用。从编译程序设计的角度来看，外体系结构的规则性、正交性、可合成性、唯一或全部原则以及提供原语而不是解法等特点是至关重要的。其实，早在1962年Burrough公司的计算机体系结构就是支持Algol式的程序设计语言的，也就是人们所常说的语言引导的体系结构。目前主要有四种情况：一是控制流体系结构，多面向命令语言；二是数据流体系结构，它是面向单赋值语言的；三是归约体系结构，它是面向应用语言的；四是逻辑推理的体系结构，它是面向谓词逻辑语言的。后三种主要是在并行计算机上支持现代的应用与程序设计语言的，而Von Neumann体系结构则不仅可

以支持命令语言，也可以用来支持说明语言。因此，考虑性能价格比等因素，未来计算机体系结构的发展能否冲破 Von Neumann 概念，目前还是一个没有解决的问题。

内体系结构研究计算机内部组织中主要功能部件的特性、它们之间的互连方式、信息流性质、以及管理这些信息流的逻辑与方法。内体系结构的目的在于设计的可理解性，以便能管理与掌握很多更低的设计细节。目前，计算机的内体系结构已经如此之多，它的分类成了一个竞相研究的课题，以便能为体系结构提供一个排序、预测与解释的基础。目前有两种分类途径：一种是按体系结构特点来分类的形态学方法；另一种是强调体系结构发展来分类的进化方法。

计算机体系结构设计的学科可以定义为支配计算机体系结构设计的规则与过程的一个系统。这样一个学科受到理论工作者与实际工作者的高度重视，并迅速地发展还是较近期的事。其原因有三：一是技术的进步，只有开发高度有序的、科学的以及逻辑的设计方法，才能管理 VLSI 体系结构的复杂性问题；二是计算机设计过程的一些重要部分已全部或部分地自动化。设计自动化的努力已经指向设计过程的最困难的自动综合阶段；三是传统的直觉的非形式体系结构设计方法不用实际物理形式已无法证明设计的正确性与精确预测体系结构的性能。因此，自 70 年代初开始，硬件描述语言已不再只是记录体系结构属性的工具，而是在几个抽象级上作为数字系统设计与实现的极端重要的部分，用来进行实验以预测与评估体系结构的性能；用来为编译程序设计者提供一个严格的规范；用来支持形式设计。美国国防部就象采用 Ada 语言作为标准程序设计语言那样，也采用 VHDL 语言作为标准硬件描述语言。

计算机体系结构与 IC 技术的发展是相辅相成的。随着特征尺寸的减小，芯片集成度是按 $O(n^2)$ 上升的，而器件的速度则是按 $O(n)$ 上升的。那么，如何选择计算机的体系结构才能使它的性能是 $O(n^3)$ 上升的呢？为此，并行计算成了研究的热点。按并行实现来说，有控制并行与数据并行两种风格；按并行粒度来说，有细粒度并行与粗粒度并行；按并行程序来说，有隐式设计与显式设计的并行。本章的目的就是要根据 IC 技术的内在特点，研究如何设计嵌入式 RISC 体系结构，才能最好地支持并行计算风格。为了形成一个系统的看法，本节首先对计算机的体系结构分类进行了研究。在 Flynn 分类的基础上扩充了分类字符并形成了一个有 8 个类别的分类方案；接着根据实现技术与应用要求的影响对 RISC 体系结构的发展进行了分析。

1.1.1 分类研究

为了对并行计算有一个系统的了解，对计算机的体系结构建立合理的分类是必要的，以便能对并行计算提供一定的解释能力与预测能力。1966 年由 Flynn 提出的分类方案采用了六个分类字符：

- IM：指令存储器；
- DM：数据存储器；
- CU：控制(或指令处理)部件；
- PU：处理(或指令执行)部件；
- IS：指令流；

• DS：数据流。

它只有一个分类层次，共有四种类别：SISD、SIMD、MISD 与 MIMD。由于 Flynn 的分类方案非常简洁，很多人就企图在分类细化方面对 Flynn 分类进行发展，但都没有取得 Flynn方案那样的成功。1990 年 S. Dasgupta 提出了一个多层次的分类方案，共用了七个分类字符及一个表示器件数目的基数(作为分类字符的下标使用)：

• iM：交错存储器；

• sM：简单存储器；

• C：隐含(cache)存储器；

• sI：简单指令准备部件/处理机；

• pI：流水指令准备部件/处理机；

• sX：简单执行部件/处理机；

• pX：流水执行部件/处理机。

它有三个分类层次，强调已有发展是如何影响体系结构的进化的，可以详细地刻划现有机器的内体系结构。不难看出，Dasgupta 方法的分类字符是以计算机内体系结构中的典型部件结构为依据的，因此，如果再出现新的典型部件结构，则这种分类方案就无法描述了。而 Flynn 的分类方法则不然，它的分类字符是以计算机体系结构中的典型逻辑概念为基础的。例如，指令存储器与数据存储器只是一种逻辑概念的区分，而物理实现上则可能就是不分开的；又例如，指令流与数据流也只是一种逻辑概念的区分，实际上这两种信息流可能是共享总线的。正是这种以逻辑概念而不是以部件结构为依据的作法带来了 Flynn 分类的简洁性、实用性与稳定性。

从解释能力上来说，Flynn 分类只用了指令流与数据流的逻辑概念，因而只能描述指令级的并行计算。由于控制并行风格的 MIMD 体系结构存在着硬件与软件上的设计困难，而数据并行风格的 SIMD 体系结构则在硬件与软件设计上都取得了一定成功。支持数据并行风格的海量并行计算机已经成为商品。著名的 SIMD 类型的 CM 机就是用硬件便于数据并行性的很好例子。在 MIMD 类型的 iPSC/2 机中通过叫做 SPMD 的数据并行处理形式可使它以近似 SIMD 的行为方式工作。这个在 Flynn 分类中没有包含的 SPMD 风格的意思是：将应用数据划分成不相交的集合，并拷贝到不同的处理元中，而且把同时处理每个数据集合的整个过程或程序段一起拷贝。每个拷贝的程序段以 MIMD 方式运行，并行地处理每个数据拷贝，直到完成它的任务。这个例子及其有关的发展暗示为了提高 Flynn 分类的解释能力，在计算机体系结构的分类方案中，程序流 PS 以及相连的程序存储器 PM 也应与指令流 IS 以及指令存储器 IM 一样作为建立分类字符的逻辑概念，以反映体系结构中进程级并行计算技术的发展。在 Flynn 的六个分类字符的基础上增加这两个分类字符之后，可以得到一个有 8 个类别的分类系统：SISD、SPSD、SIMD、SPMD、MISD、MPSD、MIMD 与 MPMD，如图 1.1 中所示。不难看出，这是通过 PM 与 PS 的引入才形成 8 个类别的。这两个概念的引入说明作为并行的单位不仅有指令而且有程序段，从而使指令级与进程(程序段)级的并行计算都能得到恰当的解释。其次，CU 与 PU 的关系已不仅是 Flynn 分类中的指令流关系，也含有交互作用的关系。同时，一个 PU 只能受一个 CU 控制，而不能受多个 CU 控制。如果真有多个 CU 控制一个 PU，则从概念上这些

CU 就可以看作一个了。如果不能看作一个 CU 的话，则说明该 PU 将接受到相互矛盾的控制信号，从而就不知如何工作了。这是没有实际意义的。反之，一个 CU 是可以控制多个 PU 的，即多个 PU 接受的是完全相同的控制信号。因此，此新分类系统中的 MISD 的图示将是与 Flynn 分类的图示不同的，而且按照新的图示 MISD 也不再只是形式类型了。共享存储器的分布式系统应当是属于这个类型的。

从预测能力上讲，以部件结构为基础的 Dasgupta 分类方法只能提供这些部件的排列组合；而以逻辑概念为基础的分类方法则仅仅提供这些概念的排列组合，而具体的体系结构则可根据这些概念组合由设计者去创造。因此，它体现了事物发展的无限性。例如，在 SISD 中 CU 可以是 sI(pI)，PU 可以是 sX(pX)；而 SPSD 中的 CU 则应是带局部存储器用以存放程序段的。SIMD 与 SPMD 则可以分别是指令级与进程级的数据并行体系结构的概念描述。MISD 与 MPSD 都可以看作是 PU 带局部存储器的分布式体系结构的概念描述，而且 MPSD 中的 CU 也是要带局部存储器的。MIMD 可以看作指令级的数据流机或请求流机的概念描述，而 MPMD 则是多微计算机系统，以及计算机局域网与宽域网等体系结构的概念描述。

1.1.2 实现技术

现代计算机主要是用高集成度的硅技术实现的。随着特征尺寸的减小，集成度是平方上升的，而器件速度是线性上升的。因此，如何有效地利用集成度按平方上升的特点是很关键的；其次，从芯片面积、功耗与速度上看，互连线的影响远远超过器件的影响。因此，通讯局部化成了 VLSI 体系结构的设计原则。灵巧存储器阵列、脉动阵列与微计算机阵列都是按这条原则设计的。

一台新计算机的设计是由该机器将如何实现与如何使用来决定的。根据上面的分析，为了能充分利用特征尺寸减小每个芯片所能提供的计算能力的 $O(n^3)$ 上升，多芯片设计(multichip design)是很难做到这一点的。一是特征尺寸的减小将会导致划分的改变，即整个处理机的重新设计；二是芯片之间的通讯将会占用 50%的时钟周期时间，当器件速度提高 1 倍时，整个多芯片机的总性能将只会提高 $(0.5/2+0.5)/1=0.75/1$，即只提高了 33%。而单片机则不然，例如，MIPS R2000 的工作频率为 8MHz，而基于同样核心设计的 R3000 的主频为 40MHz，即提高了 400%。所以，现代的 RISC 微处理机的整个设计差不多都是在同一芯片上实现的，位片式结构是无法竞争的。也正因为如此，WSI 技术受到了重视。当不能做到在同一芯片上时，芯片结构的自底向上选择也应是按这个方向进行的。

那么再如何进一步提高单机的性能呢？按照通讯局部化原则，流水线技术与 cache 技术已成了提高单机性能的两项重要硬件技术。流水线技术首先用于 Stretch 机上，是通过指令级并行性来提高性能的。目前已形成了两种 RISC 设计风格。一是使流水线更深的超流水线 (superpipeling) 风格，二是每个时钟让多条指令进入流水线的超标量(superscalar) 风格。此外，VLIW 也是一种多条指令进入流水线的机器，但它是借助编译程序寻找能同时发送的操作的，并把这些操作安排在一条超长指令字中。尽管由于倡导 VLIW 设计的 Multiflow 与 Cydrome 公司因资金不足而关闭，但这种设计思想还是有价值的。研

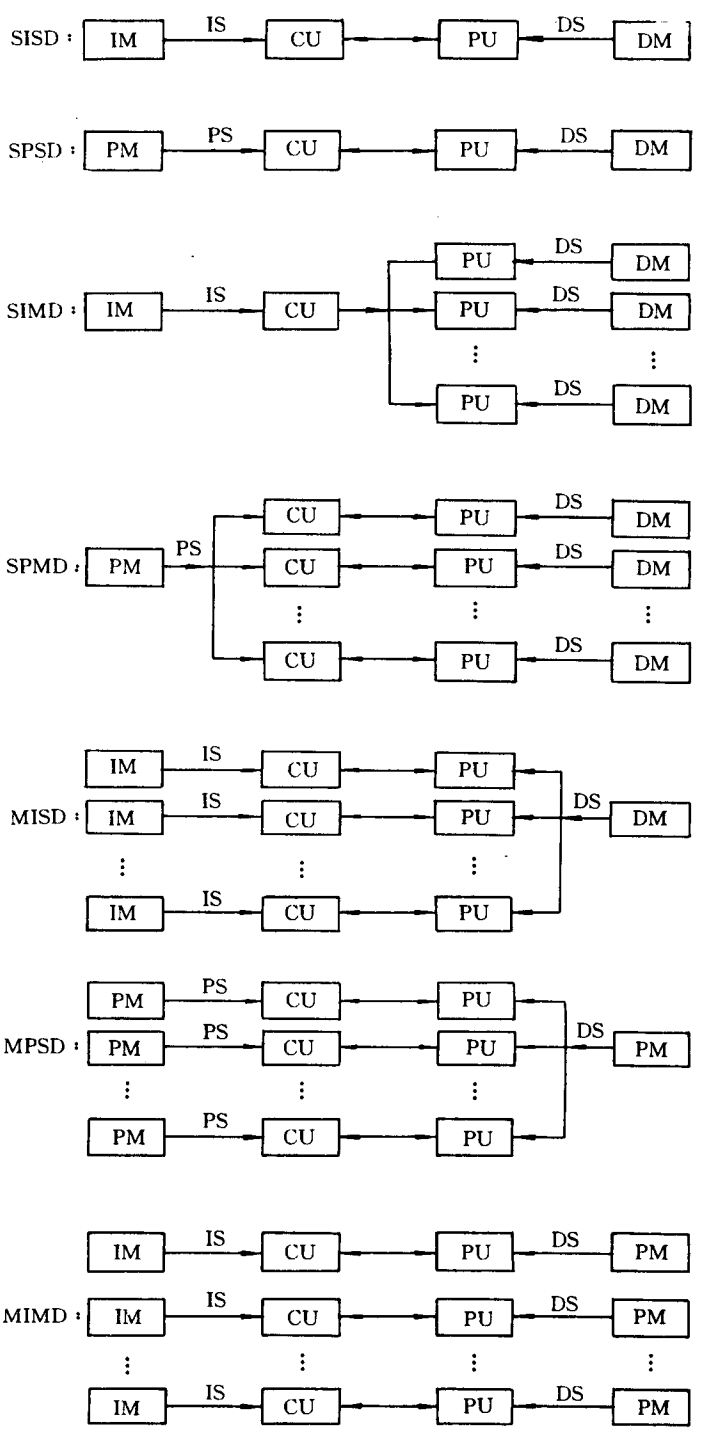


图 1.1　计算机体系结构的分类

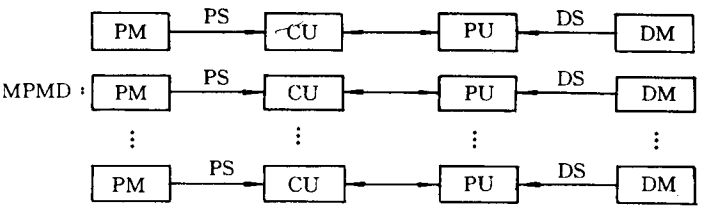


图 1.1　计算机体系结构的分类(续前页图)

究表明,同级的超流水线机与超标量机的性能是差不多的。在现有的超标量机中,都限制一个时钟周期内所能发送的指令组合。例如,两指令路的超标量机中可以让一条浮点指令与一条非浮点指令一起发送。那么,如果浮点指令在一个程序中不占 50%的话,则机器的实际性能就要下降。没有指令组合限制的超流水线机或采用动态调度的机器,可以减轻这种性能限制。因为指令级的并行性总是用这种技术提高速度的一个限制因素,流水线深度与指令发送率之间存在着一个折衷。也就是说,功能部件的延迟时间越深,指令的发送率将会下降。即减小延迟时间,而不是峰值带宽,可能会得到一个最快的机器。这种机器的典型性能改善大约为 1.5 到 2 倍。由于存储器访问周期总是慢于处理机的时钟周期,因此,在 RISC 微处理机中便采用了巨型机 Cray 的相同办法,设置了单独的存数与取数指令来提高整机的性能。不仅如此,现在的 RISC 微处理机基本都采用 60 年代末或 70 年代初大型机与小型机上的 cache 技术,可以使性能又提高 1 倍左右。目前,多 cache 技术也在研究之中。由于 RISC 微处理机在体系结构上充分利用集成度平方上升的好处,采用了巨型机、大型机与小型机的有效设计技术,在过去的六、七年中,机器性能平均每年提高 1.5～2 倍。而大型机与小型机则每年只有 25%的提高。现代 RISC 微处理机的性能已超过了大型机与小型机。芯片计算机设计时间的缩短,以及最小机器与最大机器之间的 CPU 性能差别的减小将是 90 年代为了适应 IC 技术的发展而会加速到来的两个重要结果。

从前段的体系结构分类来看,计算机性能的进一步提高就要靠多机系统了。现在的研究集中在建立使用 RISC 微处理机技术的可派生多处理机系统,从几百个处理机派生到几千个处理机等。而工业界则在建立小(4～20 处理机)的支持 cache 一致性的基于总线的机器。这种机器可以用于三个方面:一是取代分时使用的单处理机系统,使每个用户可以使用一个处理机;二是利用事物之间的并行性,可以形成良好的事物处理系统;三是可以用作科学与工程应用中的并行处理机。在这三种应用中微处理机性能的改善对充分发挥机器的有用性是很关键的。为了使程序设计容易,这些机器都设计成 cache 一致的。采用的 cache 一致性算法是靠总线传播以实现一个管理协议(snoopy protocol)的。为了减少每个处理机对总线的请求,往往是采用大的 cache 达到的。这种通过总线共享存储器的多机系统可以看作是 MISD 或 MPSD 型体系结构。因为它们共享一个单地址空间的存储器。处理机数目一般为 10 左右。当处理机的数目多得使总线饱和时,则可用开关网代替总线。这时处理机的数目大约不超过 10^3。

高集成度硅技术与科学和工程应用中的数据级并行性相结合使 SIMD 或 SPMD 型的支持数据并行计算的体系结构得到了迅速发展，相应的机器已经商品化。数据并行计算作为一种程序设计风格，它能支持由上万个甚至百万个处理机组成的机器。因此，这些机器又叫做海量并行计算机(massively parallel computers)。采用 O(N)个处理机求解大小为 N 的问题时，数据并行处理算法的典型计算时间为 O(logN)。这种机器(特别是 SPMD)的主要优点是使应用程序、编译程序以及硬件的设计简化。设计的关键是一个通用的连接各处理机的通讯网络，它使程序设计者不必详细考虑数据与硬件之间的映射问题。具有 65,536 个处理机的 CM 机器采用了可编程超立方互连网技术，数据并行语言则是由 C 与 Lisp 扩充而成的 C* 与 Lisp*。具有 16,384 个处理机的 T/4000 则是以 Transputer 微处理机作为节点控制机，而以 occam 语言为数据并行语言的。在 SIMD 机器中指令是同步执行的，而 SPMD 机器中，各处理机虽然执行的是同一程序段，但由于转移指令，同一程序段在不同处理机上的执行路径不可能是一样的。因此，需要解决同一程序段在不同处理机上执行时的同步问题。经济性使得海量并行的计算机体系结构越来越有吸引力(当然这些机器也存在开发应用的困难)。数据并行性将会象 70 年代的向量并行性一样，在 90 年代将会从秘密技术逐渐发展成主流技术。

不难看出，MISD 或 MPSD 类型的多处理机系统显然存在着 von Neumann 机固有的处理机与存储器之间的瓶颈问题。而 MPMD 类型的多计算机系统通过处理机与存储器的局部化以及消息传递的进程之间的通讯，则可以用来建立具有上百万个节点的多计算机系统。这种系统的设计空间是消息传递网与节点计算机的不同组织。例如，节点计算机可以是单计算机或多处理机。因此，这种机器的设计空间是很大的。最早的商用多计算机系统，例如 Intel iPSC/1，Ametek S/14 以及 N-cube/ten 等的组织、系统软件与程序设计方法都是源于 1981 年的 Cosmic cube 机。

1.1.3 应用要求

从应用要求来看，通用计算与科学计算程序的差别是显著的。在科学计算中存在着大量的数据级并行性，例如矩阵计算等。并行性是与矩阵的大小，即数据的多少成比例的。这些计算程序至少在部分程序代码中实质上具有无限的并行性。实现这种并行性的最好方法是什么，目前还不清楚，除了 RISC 中的多指令执行外，向量机与 SIMD 机提供了实现这种并行性的重要方法。对于通用计算来说，有关标量计算程序的研究表明可利用的并行性是每个时钟周期大约两条指令。

从应用领域上来说，数据通讯、并行计算与人工智能将对计算机体系结构产生新的影响。数据通讯与计算机过去是彼此独立发展的。然而很久以来人们就提倡完全集成的各种计算机网络系统。这种计算机系统至少有下列五个研究问题：并行粒度，同步机构，存储器等待时间，互连网络以及管理方法。宽域网与局域网的发展趋势已非常明显，那么并行计算问题能不能也在控制流体系结构的基础上采用通讯网的概念来突破呢？这是通讯界与微电子学界对未来计算机的发展观点。

在多处理机中为了保持指令之间的执行顺序与数据依赖关系而必不可少的调度与同步开销使细粒度并行计算的效率不高。从概念上看，数据流机是一种很好的支持细粒度并

行的非 Von Neumann 体系结构。但是它要求有新的数据流语言来支持。这种语言必须使程序员能给出无副效应的计算；必须使其编译程序很容易地检测数据流程序中操作之间的数据依赖性，以便产生数据流图。日本的第五代计算机计划是支持数据流机的。另一种支持细粒度并行的非 Von Neumann 体系结构是归约机。它是请求流驱动的。有串归约与图归约两种归约方法。据估计，下一代计算机将需要具有 10^8 到 10^9lips (logical inferences per second) 之间的速度。也就是 10^4MIPS 与 10^6MIPS 之间的计算速度。采用怎样的体系结构：控制流、数据流、还是请求流才行呢？这是目前尚未解决的难题。

人工智能一般认为是从 1956 年开始的。60 年代开始的以计算机为核心的机器人，第一次全面地模拟和延伸人的感觉器官、响应器官与思维器官。从 70 年代起人工智能的研究进入了专家系统的新阶段，在模拟人的推理能力上前进了重要的一步。人工智能的目的是什么？目前的回答还是不统一的。第一种是使计算机智能化，成为更有用的机器；第二种回答是利用计算机模拟人的行为；第三种回答是研究整个智能事物的空间。在人类历史上人工飞行的研究揭示了自然飞行的空气动力学奥秘。那么，人工智能的研究能不能揭示自然智能的思维过程的奥秘呢？攻克这个问题的办法有两种：一是全面的开展研究；二是从比较小的容易取得成果的问题开始（例如专家系统）。我们将按后一种办法，从支持专家系统的智能协处理机开始，研究智能计算机问题。

1.2 编译程序与 RISC 体系结构

现在，大多数程序都是用高级语言设计的，指令集合实质上是一个编译程序的目标语言。所以，编译程序对计算机的性能将有很大的影响，了解今天的编译程序技术对设计与有效地实现 RISC 体系结构是关键的。换句话说，RISC 体系结构的选择将影响机器所能产生的程序质量，以及建立一个好的编译程序的复杂性。这一节将主要从编译程序的角度讨论指令集合的关键之点。为什么只讨论计算机的体系结构而不讨论实现呢？因为与过去的 CISC 芯片选择不一样，选择 RISC 芯片并打破与现有生产线的用户软件兼容性应成为设计的主要决定。由于工业标准 ABI (Application Binary Interface)的出现，此标准将支持被编译的应用程序能在基于同样 CPU 体系结构的计算机系统上运行，应用软件的兼容性变得越来越重要。可用的用户应用软件包的数量在继续增长，因此，改变一个体系结构变得非常困难，而改变一个实现(内体系结构)则变得越来越容易。

一般说来，改变一个体系结构意味着用户的应用软件将必须改变。由于计算机厂商不是自己写所有的应用程序，而且大量软件包必须修改，这种改变的代价是非常高的。在某些情况下，体系结构是以用户应用软件能前后兼容的方式修改的。但一般说来，这是难办到的，所以，选择一个好的体系结构是很关键的。

实现的改变意味着只改变硬件与可能必要的操作系统软件。因为厂商是同时调整硬件和操作系统的，所以，实现中的开销同用户软件改变比较起来还是小的。一个给定实现中的任何限制可以(通常是)在下一次实现中消除或克服的。所以，基于 RISC 的一个给定实现的选择不是很关键的，最重要的选择准则是体系结构，包括它的当前实现与未来实现，而不仅仅是体系结构的当前实现。

许多RISC体系结构的评价是先做性能比较，再在此基础上评价体系结构。这种方法所得到的是实现的特点，而不是体系结构本身的特点。例如，有些评价认为i860，88000与SPARC有表1.1中所示的体系结构缺点。事实上这些都是实现上的特点，而不是体系结构的特点。例如，外总线的数目虽然是RISC实现性能的主要成份，但不是体系结构的特点，可以在不影响体系结构的情况下，从一个实现到另一个实现时改变外总线的数目、速度与宽度。同样，Cache一致性在多处理机系统的实现中是一个关键特点，但也不是一个体系结构的特点，在不影响处理机体系结构的情况下，可以加进，删除或改变Cache一致性。归纳上述，本节将只对RISC的体系结构进行讨论与选择。

RISC的设计哲学就是简单性与有效性。也就是说RISC是通过明智的简化处理机指令集合的语义以及指令集合的编码来获得芯片资源的有效利用的。RISC的关键特点是：

表1.1 所谓体系结构缺点

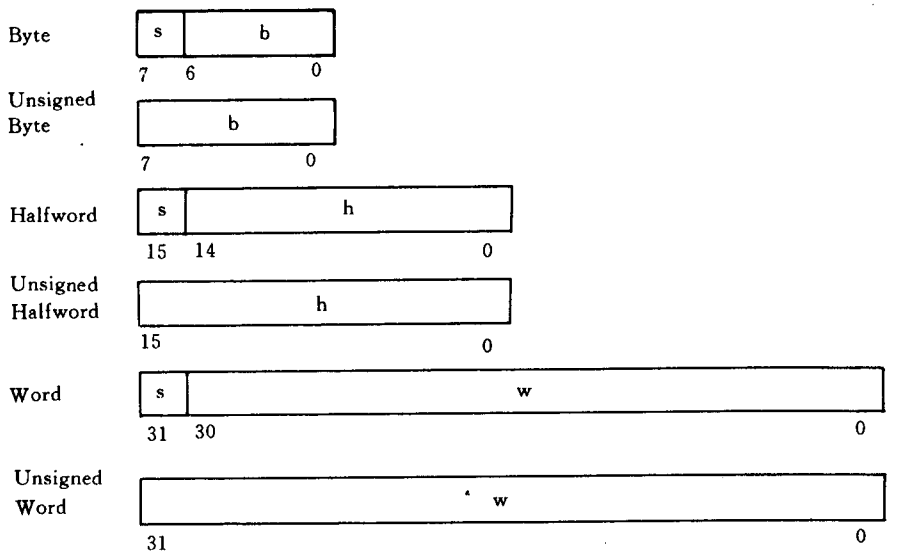
体系结构	缺点
i860	No cache conherency for internal caches
88000	No dual-cache tags Only supports MESI model
SPARC	Single address/data bus No separate address adder

- 单周期操作(对于大多数指令)；
- 分开的存取指令；
- 基于寄存器的执行；
- 固定格式与固定长度的指令；
- 简单的寻址方式；
- 大的寄存器集合或寄存器窗口；
- 以及延迟转移指令等。

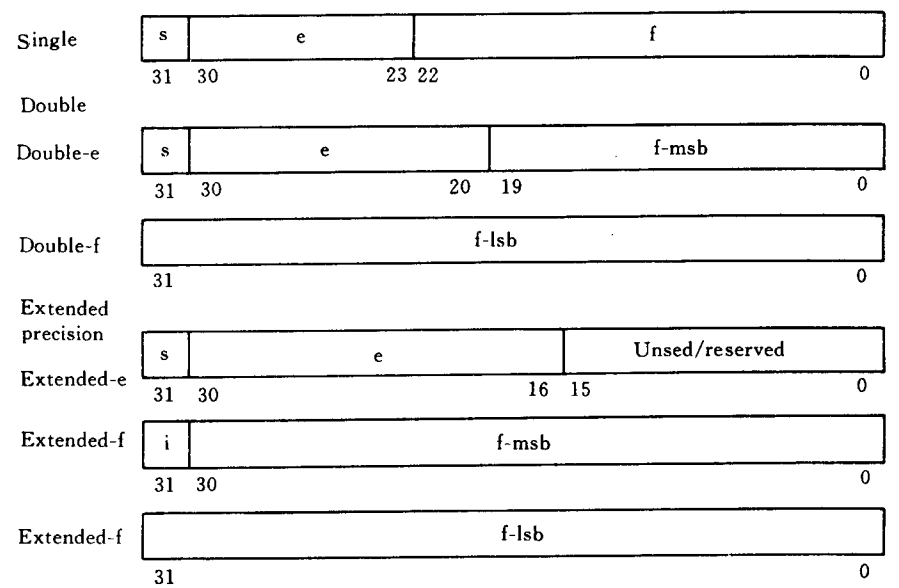
因此，RISC体系结构可以从数据类型、寻址方式、寄存器模型、存储器管理以及指令集合等五个方面进行讨论与选择。

1.2.1 数据类型(data type)

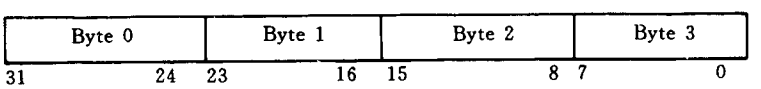
数据类型的使用情况可以用来决定哪些类型的有效支持是最重要的。图1.2(e)表示一种C编译程序GCC，SPICE程序以及一种C版本的文档处理程序TeX使用数据类型的情况。可以看出，对这些程序32位与64位的数据是重要的。故所有的RISC体系结构都应支持定点数数据类型，也就是字节，无符号字节，半字，无符号半字，字，以及无符号字。如图1.2(a)所示。所有RISC体系结构也都应支持ANSI/IEEE浮点数数据类型，也就是单精度与双精度浮点数，如图1.2(b)所示。从字长上看，从32位到64位的发展已很明显。因此，在选择嵌入式RISC的体系结构时，将考虑它是同时支持32位与64位微处理机的实现的。当采用64位的微处理机结构时，64位的长字当然也是要包含在定点数据



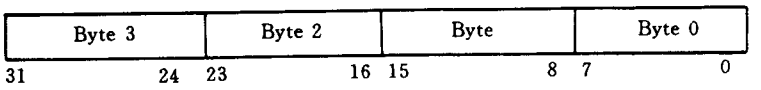
(a) 定点数数据类型



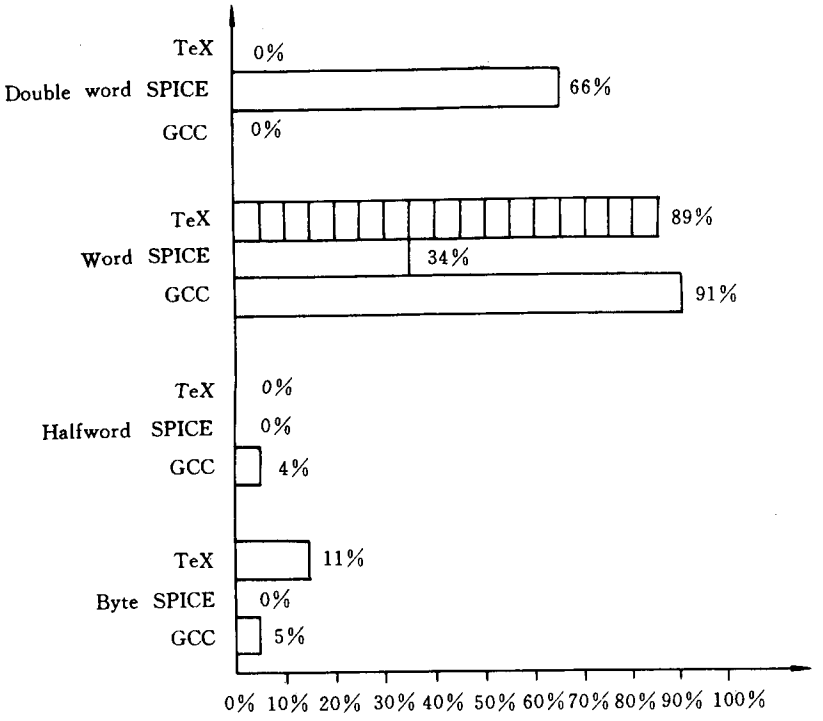
(b) 浮点数数据类型



(c) 32 位字的大端次序



(d) 32 位字的小端次序



(e) 数据类型使用情况

图 1.2 数据类型

类型之中的。尽管现在的语言标准不支持扩展精度的浮点数，但由于 RISC 的实现已达到大型机(mainframe)的性能，对扩展精度浮点数的要求将会上升。这种数据类型的表示如图 1.2(b)所示，SPARC 体系结构是支持它的。

字节次序或者说寻址方案有大端次序(Big-endian Ordering) 与小端次序(little－endian Ordering)，如图 1.2(c)与(d) 所示。它建立往微计算机送数的格式。大端格式则首先送最低有效位字节。RISC 体系结构对字节次序的选择主要是根据兼容性原则出发的。SPARC 支持大端次序，因为基于 Motorola 680X0 的大多数产品是采用大端次序的。虽然 i860 与 88000 支持两种字节次序，能为无论那种字节次序的机器提供接口方便，但 88000ABI 指定大端格式为操作系统的接口；i860 ABI 将指定哪种字节次序尚不清楚，但如果为与 Intel80X86 兼容，可能要采用小端格式。

除了上述基本的数据类型外，为了支持不同的应用，占领不同的目标市场，RISC 体系结构往往还支持一些附加的数据类型。例如，i860 支持 8 位、16 位与 32 位象素(pixel)以

提供高性能的三维图示处理;88000 支持一个字内的位场数据(bit-field data),可以支持比具有跨字界位场指令的 Motorola 68020 较窄范围的字符串处理应用; SPARC 支持表征数据 (tagged data), 这对于采用动态数据类型的 (例如 Smalltalk 语言) 系统可节省 20～25%的执行时间。因为特殊数据类型是针对特殊应用的,对通用计算这些数据类型及其有关操作不会带来多大的性能影响。顺便指出,数据类型有两种表示方法。一是常用的通过操作码的编码来表示。二是通过数内部表征位(tags)来表示。这些表征位指明被操作数的类型,根据它执行相应的操作,这种体系结构的机器很少,Barrough 公司的机器是这方面最突出的例子,还有 Symbolics 公司也是用这种体系结构设计 LISP 机的。

1.2.2 寻址方式(addressing mode)

寻址方式涉及到如何解释存储器地址以及如何指定存储器地址。如何解释就是指存储器地址访问的对象及其长度是什么? 所有机器差不多都是访问字节的,并能提供字节(8 位),半字(16 位),以及字(32 位)与长字(64 位)的访问。如上段所述,字节有大端与小端两种次序,这里将采用小端次序。为了使实现简单将约定长于字节的访问是对准的。如何指定存储器地址,也就是寻址方式的选择。寻址方式具有大量减少指令数目的能力,但也增加芯片设计的复杂性,因此需要合理地选择。RISC 体系结构一般都支持最常用的简单寻址方式,以优化程序密度与速度。如表 1.2 所示,最常用的简单寻址方式有: 立即值方式、寄存器方式、寄存器间接方式、变址值方式、位移值方式与相对地址方式等。

表 1.2 RISC 体系结构的常用寻址方式

方　式	描　述
immediate	Value, signed and unsigned
register	Rx, x 是寄存器编号
register indirect	(Rx), x 是寄存器编号
register indirect with index	(Rx, Ry), x,y 是寄存器编号
register indirect with displacement	(Rx)+displacement, x 是寄存器编号
PC with displacement	(PC)+displacement, PC 是程序计数器

立即寻址方式就是利用指令格式中给出的值,直接作为寄存器操作的被操作数,对于立即寻址方式需要确定的有两点。一是立即值的大小。i860 体系结构支持算术操作的 16 位有符号立即方式,逻辑操作的 16 位无符号立即方式; 88000 支持 16 位无符号立即方式; SPARC 体系结构仅支持 13 位有符号立即方式。由于长立即方式用到很少,立即方式的长度是没有关系的。然而,有符号的立即方式比无符号立即方式提供额外的灵活性。二是立即值的使用频率。统计表明它在取数指令,比较指令以及 ALU 操作指令中的使用频率都是很高的。因此,是一种不可缺少的寻址方式。

寄存器方式用来指定寄存器的编号。RISC 体系结构大都采用寄存器操作的三地址方式。因此,在寄存器操作的指令格式中要安排两个源地址与一个目的地地址。

寄存器间接方式允许地址是由寄存器的内容指定的。

变址值方式就是对寄存器内容加上一个变址值。据统计，这种方式仅有 6%的使用率。它对人工智能语言与科学计算是有用的，对于一般计算影响不大。88000，i860 的存取指令不支持这种方式。

位移值方式就是对寄存器内容加立即位移值。位移值的大小是最难解决的，统计表明由于变量的多个存储区域，需要用不同的位移值来访问它们，位移值的分布是很宽的。但太大的位移值又影响指令的长度。i860 88000 与 SPARC 分别支持 16 位有符号位移值，16 位无符号位移值，以及 13 位有符号位移值方式。然而，有符号的立即位移值比无符号的立即位移值更灵活。

相对地址就是对程序计数器加位移值，是转移指令中常用的地址方式，所有 i860、88000 与 SPARC 体系结构都提供两种寻址方法：PC 相对寻址与寄存器间接寻址。对于 PC 相对寻址，i860 提供 16 位与 26 位的位移值方式；88000 提供 16 位的位移值；而 SPARC 则提供 22 位的位移值。根据 CISC 机器上的研究，93%的 PC 相对转移只用到 16 位的位移值方式；87%的 PC 相对转移只用到 15 位的位移值方式就够了。但在 RISC 机上由于程序要长些，16 位的位移值大概能覆盖 87%的 PC 相对转移。

寻址方式的如何编码依赖于寻址方式的多少以及操作码与方式之间的独立程度。对于少量的寻址方式，即操作码与寻址方式的组合很少的情况下，可以由操作码来隐含地指示。而组合较大的情况时，则在指令格式中需要加单独的寻址方式场表示。在 RISC 机中寻址方式一般都很少而不用单独的寻址方式场指示。

1.2.3 寄存器模型(Register model)

指令被操作数在 CPU 中有堆栈、累加器与寄存器三种存放形式。堆栈型的好处是有简单的逆波兰表示的表达式计算模型，代码密度高，指令字短，例如，Transputer 微机的指令长度只有 8 位。缺点是堆栈不能随机访问，这个限制使它难以产生有效的程序代码。因为堆栈是一个瓶颈，也难以有效地实现。累加器型的好处是机器的内部状态极小，指令短。缺点是因为只有一个暂存寄存器，存储器传送将是很多的。但如果存储器与 CPU 能作到同一芯片上时，这个缺点将不是严重的。在第六章智能 RISC 中我们将采用这种体系结构。寄存器型的好处是代码产生有最通用的模型，缺点是所有被操作数必须是命名的，因而指令较长。现在的 RISC 差不多均采用寄存器型的体系结构，理由有二：一是寄存器比存储器快，二是编译程序更容易有效地使用寄存器。所以，这里将只讨论寄存器模型的 RISC 体系结构。RISC 体系结构的寄存器模型取决于寄存器的多少及其管理方式。

究竟多少个寄存器是充分的？这依赖于编译程序如何使用它们。为此，可考察一下现代高级语言的三个分开的数据区域。首先是堆栈区，用来安排局部变量。此堆栈随着过程调用与返回而增加与减少。堆栈上的对象是相对于堆栈指针访问的，而且主要是标量(简单变量)，而不是数组。此堆栈是用作活动记录的，而不是计算表达式的堆栈。所以，差不多从来不是弹出与压入堆栈的。其次是总体数据区，用来安排静态说明的对象，例如整体变量与常数。这些对象中的大部分是数组或其它集总的数据结构。第三是堆区，用来安排不遵守堆栈原则的动态对象。堆区中的对象是用指示字访问的，通常不是标量。在上述三

个数据区中，只有堆栈区的数据最适合安排到寄存器中，而堆区的数据是用指示字访问的，是不可能安排到寄存器中的。整体变量与某些堆栈变量是有别名的，即可以多种路径访问时也是不能安排到寄存器中的。图 1.2(e)中的三个程序的执行情况表明，无论是整数寄存器，还是浮点寄存器，32 个的数量是充分的。而且现在的编译程序还不能使用更多的寄存器，也许将来不是这样。现有的一些 RISC 体系结构的可用寄存器数目的情况，如表 1.3 所示。88000 只有 32 个 32 位的寄存器，用于整数与浮点数据操作。由于每个浮点数要占用两个寄存器，所以能存储的被操作数的数目远小于 32 个。而 i860 与 SPARC 则对整数操作与浮点数操作各分别有 32 个 32 位的寄存器。研究表明，这些增加的寄存器数

表 1.3　一些 RISC 体系结构中的寄存器数目

Larger register file (>32)	
Window management policy	
RISC Ⅱ	138
MIRIS	2048
MULTRIS	1024
Pyramid	528
SPARC	128
No windows	
Intel 80960	84
AMD 29000	192
Metaforth MF1600	1024
Moderate register file (32)	
IBM 801	32
MIPS	32
Intel 80860	32
Motorola 88000	32
PRISM	32
HP Precision Architecture	32
Small register file (<32)	
Ridge	16
Acom—VLSI technology	16
Transputer	6
IBM RT	16
Clipper	16

目更好地提高了它们的性能。所以，少于 32 个的情况对今后的发展来说是不可取的。

对于大的 CPU 寄存器文件，其好处是：首先可以通过减少 CPU 与存储器之间的传送而加快操作速度；其次在 CPU 内支持过程参数的传递；第三在 CPU 内支持多任务上下文转换与中断处理；第四芯片上的数据堆栈与/ 或队列；第五提高芯片的规则度。大 CPU 寄存器文件的缺点是：首先是更长的访问时间；其次如果采用窗口指示字，寄存器地址译码时间更长；第三寄存器文件占用更多芯片面积；第四窗口策略越灵活，CPU 逻

辑越复杂；第五对于相对小些的寄存器文件，先进的编译技术更有效；第六如果 CPU 寄存器要根据上下文转换来保存，则寄存器文件越大用的存储时间越多。

从管理方式上讲，有 Window 模型、Cache 模型与 Matrix 模型。SPARC 体系结构是支持 Window 模型的，它可以提供 2 到 32 个窗口，为了提供无限的窗口深度，并组织成循环堆栈的形式，如图 1.3 所示。窗口模型的赞成者认为它有下列优点：首先是能提高函数

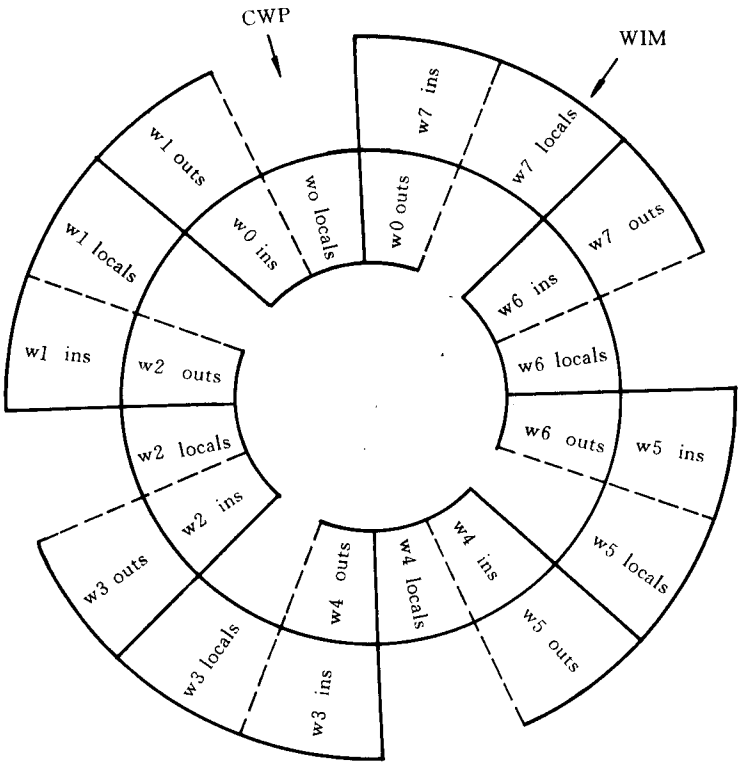


图 1.3　窗口寄存器的循环堆栈

调用的速度，编译程序不必为函数调用而存取寄存器；其次是编译程序不必完成复杂的寄存器调度而变得简单；第三是窗口系统可以用户软件透明的方式增加窗口数目。而窗口模型的反对者则认为它有下列缺点：首先是循环堆栈的上溢出与下溢出处理，其次是上下文转换比传统分配方式要求存取更多寄存器的功能。

i960 体系结构支持寄存器的 Cache 模型。用户直接访问 32 个 32 位的通用寄存器与 32 个专用功能寄存器(SFR)，如图 1.4 中所示。SFR 寄存器对芯片上的外设器件提供实时寄存器接口(这些寄存器的内容不是体系结构定义的)用来管理特定实现的硬件。32 个通用寄存器中 16 个是整体寄存器，另 16 个是局部寄存器。当过程调用时，16 个整体寄存器仍然是可见而不改变的，体现其它体系结构“正常”寄存器的特点。调用指令 CALL 使局部寄存器隐含地更换，而返回指令 RET 自动恢复它们。当发生一个过程调用时局部寄存器中的数据自动移到一个寄存器 Cache 中。于是，不要求被调用过程明显地保留局部寄