1.1 计算机组成与体系结构

在描述计疗机系统时， 通常要区分计算机体并结构和计算机组成这两个基本慨念。 虽然很难给出这两个术诸的精确定义， 但对 它们所涉及的领域存在轩共识（见文献[ VRAN80) 、 [SIEW82)和[BELL78a]}, 一种打趣的可供选择的观点可参见文献[ REDD76) 。

计算机体系结构是那些对程序员可见的系统属性， 换句话说， 这些展性直搂影响到程序的逻耜执行。 计算机组成是实现结构规范的操作单元及其相互正接。 例如， 体系结构的属性包括指令集 ， 用来表示各种数据类型（例如， 数据 ， 字符）的比特数、 愉入输出机制以及内存寻址技术 组成的属性包括那些对程序员可见的硬件细节 ， 如控制信号 、 计伴机和外设的搂口以及存储器使用的技术。

例如 ， 计符机是否有乘法指令是体系结构设汁的问题。 而这条指令是由特定的乘法单元实现，还是通过顶复使用系统的加法单元来实现， 则是组成的问题。 组成基于乘法单元使用的预期频度 、 两种方案的相对速度以及特定乘法单元的成本和物理尺寸笘因素。

无论是过去还是现在， 了解体系结构与组成之间的差别都是很项要的。 很多计钉机制造商会提供系列机产品 ， 它们打籽相同的体系结构， 但组成是不相同的， 因此， 同一系列中不同型号的计杆机的价格和性能也不相同。 进一步来说， 一种特殊的体系结构可以存在多年， 井且覆盖多种不同的计符机型号， 但它的组成则随疗技术的进步而不断更新。 这种现象的一个突出例子怂IBM Sys<em/370体系结构， 这种架构于1970年推出， 包括多种型号。 低需求的客户可以购买较便宜、 速度较慢的型号， 如果今后要求提环了 ， 可以升级到更员的、 速度更快的型号， 而不必丢弁巳经开发的软件。 近儿年， IBM通过改进技术推巾了许多新型号来替代旧的型号， 为用户提供 称速， 低价或两者兼备的产品。 这些新型号保WI了同祥的体系结构， 因而保熬了用户的软件资税 值得注意的足， s,,.,m1,10体系结构经过儿次书强 ， 不但生存至今、 而且仍是IBM的旗舰产品。

在被称之为微计疗机的一类计n:机系统中， 体系结构和组成的关系非常紧密。 技术的更新不仅影响了计算机的组成， 而且还导致了更强大和更复杂的体系结构。 通常， 越小的机器， 新旧 两代之间的兼容性要求越少， 因此组成和体系结构设计决策的关系就更加紧密。 关于它的一个 打趣的例子就是精简指令集计算机(RISC), 本书将在第13章进行深人的讨论。

本书介绍计打机组成和计算机体系结构两个方面的内容， 或许更强询组成方面的内容。 但是，计算机组成的设计必须辽昭特定的体系结构规范， 因此 ， 对组成的深入论述也要求对体系结构打同样细致的考察。

1.2 结构和功能

计杆机是一个复杂的系统， 当代计算机包含数百万个电子元件， 那么 ， 怎样才能i,Y楚地描述它们呢？关键就在于认识包括计n:机在内的大多数复杂系统的分层性质[SIM096] 。 层次系统是一系列互相关联的子系统， 每个子系统又在结构上分层， 仇到分成我们所能达到的一些基本子系统的及低级。

复杂系统的层次特性是设计和描述它们的基础。 设计者每次只需要处理系统的某个特定层次即可。 在每一层中， 系统由一组部件及其相互关联所组成。 每一层的行为仅仅依赖于系统下一层更为简单的抽象特征。 在每一层上， 设计者关心的是结构和功能。

• 结构：部件相互关联的方法。

• 功能：作为结构组成部分的单个独立部件的橾作。

根据描述 ， 有两种选择 由底层开始， 向上建立完善的描述；或者从顶层开始， 将系统分解成各个子部分。 许多领域的邓实证明 ， 自顶向下的方法是最消晰且砓有效的方法[WEIN75]。

本书采用的方法也遵循这一观点 ， 将自顶向下地描述计杆机系统。 从系统的主要部件开始， 描述它的结构和功能， 然后还级深人推进到层次结构的底层。 本节的其余部分将为这种逐级推进的描述提供简短的概述。

1.2.1 功能

从本质上来说， 计算机的结构和功能都很简单。概括起来计算机所能执行的基本功能包含4项：数据处理、 数据存储 、 数据传送、 控制。

数据处理：数据可以有多种形式， 处理的要求也很广泛。 但是我们将行 到数据处理的基本方法或类型只打儿种。

数据存储：即使计打机最简单地处理数据（例如 ， 数据输人并处理、 结果扛接输出） ， 它至少也必须在某个特定的时刻临时存储它正在运n的数据值。 因此 ， 计算机至少要有短期数据存储的功能。 汁扦机长期存储数据的功能也同样项要。 存储在计算机中的数据文什可以用于以后的检索或更新。

数据传送：计符机的操作包含作为数据OO或者目的地的设备当数据从立接与计算机相连的设备中发送或接收时， 这个过程被称为输入给出(VO), 而这个设备被称为外围设备(periphe,,I) 。 当数据传至更远处， 或从远方设备接收时， 这个过程称为数据通信。

控制：在计算机内部， 控制器根据这些指令忤理计n机的资源， 并协词各个功能部件的性能。

前面的讨论似乎过于概括， 但即使在计算机结构的最高层次， 区分许多不同的功能仍是可 能的。 这里引用文献[SIEW82]中的一段话. "为适应功能而改变计”机的结构的情况很少发生 计算机的通用性是根本， 所有的功能专门化均发生在编程阶段 ， 而不是设计阶段。 ”

We now look in a general way at the internal structure of a computer. We begin with

a traditional computer with a single processor that employs a microprogrammed

control unit, then examine a typical multicore structure.simple single processor

computer Figure 1.1 provides a hierarchical view of the internal structure of a traditional single processor computer. There are four main structural components:

我们现在以一般的方式来看待计算机的内部结构。 我们开始

带有单个处理器的传统计算机，采用微程序设计

控制单元，然后检查典型的多核结构。

简单的单处理器计算机图1.1提供了一个分层视图

传统的单处理器计算机的内部结构。 有四个主要的结构组件：

我们先以常规方式来看一下计算机的内部结构，从单个处理器的传统计算机开始

中央处理单元（CPU）：它控制计算机的操作并且执行数据处理功能，通常减淡地被称为处理器。

主存储器：存储数据

I/O：在计算机及其外部环境之间传输数据。

系统互连：为CPU、主存储器和I/O之间提供一些通信机制。系统互连常见的例子是利用系统总线， 它由一系列导线组成， 所有其他组件连接在导线上。

上述各种组件可能有一个或者多个， 但传统上， 处理器仅仅只有一个。 近年来， 在单机系统中采用多个处理器的情况越来越多。 随疗内容的深入， 本书会涉及和讨论有关多处理器系统的 设计问题， 第五部分将专门讨论这样的系统。

第二部分将详细地考察以上每个组件。 但对我们来说， 最有趣的、 在某种程度上也是最复杂 的组件是 CPU, 其主要 构组件如下所示：

• 控制单元：控制结CPU 以至于整个计算机的操作。

• 算术逻辑单元(ALU), 执行计算机的数据处理功能。

• 寄存器．提供 CPU 的内部存储。

• CPU 内部互连· 提供控制特、 ALU和寄存器之间的某种通信机制。

第三部分会讨沦以上各个部件． 我们将行到使用并行和流水线技术所带来的复杂性。最后． 实现控制器的方法有多种． 一种常用的方法是微程序的实现技术。 从根本上讲， 微程序控制器通过执行一系列的饭指令来操作，而微指令定义了控制器的功能。使用这种方法，控制器的结构可以描述为图1-1所示，本书在第四部分讨论了这种结构。

多核计算机结构 如上所述，现代计算机通常具有多个处理器。 多核计算机即单个芯片有多个处理器，每个处理单元（包括控制单元，ALU，寄存器以及高速缓存）称为一个内核。 以下定义将解释这些术语。

中央处理单元（CPU）：计算机中提取和处理指令的部分，它由一个ALU，一个控制单元和寄存器组成。在单处理单元的系统中，我们通常称他为处理器。

内核：处理器芯片上的独立处理单元。内核在功能上与单处理器上的CPU相当，其他的特殊处理单元，比如针对矢量和矩阵运算优化的处理单元，也被称为内核。

处理器：包含一个或多个内核的物理硅片。它是解释和执行指令的计算机组件。如果一个处理器包含多个内核，则称为多核处理器。

经过大约十年的讨论，业界对这种说法普遍认可。

现在的计算机另一个显著的特点是在处理器和主存之间存在多层级存储器，我们称为高速缓存存储器，在第四章将会做详细介绍。本节我们简单介绍了高速缓存存储器比主存储器更小且更快，并通过放入主存的缓存数据加速存储器访问，这很可能在不久的将来使用在不久的将来可能将会被使用

通过多级缓存性能可以得到很大提升，其中1级缓存（L1）离内核最近，L2，L3等级别逐级远离内核，在此设定下，n级比n+1级更小且更快。

图1.2是标准多核计算机主要部件的简化视图。大多数计算机都安装在一个主板上，比如智能手机中的嵌入式系统，平板电脑，个人电脑，笔记本和工作站等。在介绍这种配置之前，我们先来看一些术语的定义。

印刷电路板（PCB）是一块刚板，用于固定和互连

芯片与其他电子元件。 PCB由层组成，通常是两

到十层，这些层通过铜路互连组件构成一块板互。 计算机中的主要印刷电路板称为系统板或主板，而插入主板插槽的较小的板称为扩展板。

主板上最重要的部件是芯片，芯片是一块半导体硅晶片，在其上制造电子电路和逻辑门，就产生了集成电路。

Next, we zoom in on the structure of a single core, which occupies a portion of

the processor chip. In general terms, the functional elements of a core are:

主板上有处理器芯片的插座或插槽，这种处理器通常包含多个独立内核，被称为多核处理器。除此之外，主板上还有存储芯片，I/ O控制器芯片和其他重要计算机部件的插槽。对于台式机来说，扩展板上的扩展插槽可以包含更多组件。因此，现在的主板只能连接几个独立的芯片组件，其中每个芯片含有几千到亿万个晶体管。

图1.2可以看到一个包含8个内核和L3缓存的处理器芯片。

但没有表明内核和缓存以及内核和主板上的外部电路之间的控制操作所需的逻辑

。该图表明L3高速缓存占用了芯片表面的两个不同部分。

但是通常情况下，所有内核都可以通过前述控制电路访问整个L3缓存。图1.2所示的处理器芯片并不代表任何特定的产品，只是提供了这种芯片布局的一般思路。

接下来我们单独研究一个内核的结构，它是处理器芯片的一部分。一般而言，内核的功能结构包括：

指令逻辑：提取并解码指令以确定指令操作以及操作数的存储地址

算术逻辑单元（ALU）：执行指令指定的操作

加载/存储逻辑：管理主存的数据传送（Cache???P8）

内核还包括一个一级缓存（L1），L1可分为指令缓存和数据缓存两部分，一级指令缓存用来传输主存的指令，一级数据缓存用来传输操作数和结果。现在的处理器芯片的内核通常还包括一个二级缓存，这个二级缓存一般也分为指令和数据缓存两部分，虽然这是一个整体，但也可以单独使用二级缓存。

大家要知道内核结构的这种表述只是提供了一个一般思路。在给定的产品中，它的功能结构未必是按照图1.2给出的三部分布局那样，特别是当这些功能的部分或者全部作为微程序控制单元的一部分时。

例子 我们来看一下现实生活中展示了计算机层次结构的例子可能会有所启发。图1.3是围绕两个英特尔四核至强处理器芯片构建的计算机主板。本书随后将讨论图1.3中标注的其他部分，现在我们先来看一下除处理器插槽外最重要的部分：

用于高端显示适配器和其他外围设备的PCI-Express插槽

（3.6节介绍PCIe）。

以太网控制器和以太网端口用于网络连接

用于外围设备的USB插槽

串行ATA（SATA）接口用于连接到磁盘存储器（第7.7节

讨论以太网，USB和SATA）。

DDR（双倍数据速率）主存储器芯片接口（5.3节讨论

DDR）。

英特尔3420芯片组是用于外围设备和主存储器之间直接访问的I / O控制器

（第7.5节讨论DDR）

下面我们自上而下的进行说明，从图1.1和1.2中可以看到处理器芯片的内部结构，为了多样化期间，我们先不研究英特尔的处理器芯片，来看一下IBM的芯片，图1.4是IBM zEnterprise EC12大型机的处理器芯片，该芯片有27.5亿个晶体管。叠加的标签表示如何分配硅片构成芯片。我们看到这个芯片有六个内核或处理器。此外，还有两个标有L3缓存的大块区域，由六个处理器共享。 L3控制逻辑控制L3高速缓存与内核之间以及L3高速缓存与外部环境之间的流量。此外，内核和L3缓存之间还有存储控制（SC）逻辑。存储控制器（MC）控制对芯片外部存储器的访问。 GX I / O总线控制访问I / O的通道适配器的接口。

再向下深入一层，我们研究内核的内部结构，如图1.5所示，这是构成单处理器芯片的硅表面区域的一部分。这个内核区域的主要分区如下：

ISU（指令序列单元）：确定超标量体系结构（第16章）中指令的执行顺序。

IFU（指令获取单元）：获取指令的逻辑

IDU（指令解码单元）：IDU处理IFU的结果，并负责所有z / Architecture操作代码的解析和解码。

LSU（加载/存储单元）：LSU包含96 kB L1数据缓存1，并管理L2数据缓存和功能执行单元之间的数据通信。它负责处理z / Architecture中定义的所有长度，模式和格式的所有类型的操作数访问。

XU（翻译单元）：该单元通过指令把逻辑地址

转换为主内存中的物理地址。 XU还包含一个快表（TLB）用于加速内存访问。第8章将会讨论TLB。

FXU（定点单元）：FXU执行定点算术运算。

BFU（二进制浮点单元）：BFU处理所有二进制和十六进制浮点运算以及定点乘法运算。

DFU（十进制浮点数单元）：DFU处理十进制数的定点和浮点操作。

RU（恢复单元）：RU保留包含所有寄存器的系统完整状态的副本，收集硬件故障信号并管理硬件恢复操作。

COP（专用协处理器）：COP负责每个内核的数据压缩和加密功能。

I-cache：这是一个64 kB L1指令缓存，允许IFU在需要时预取指令。

L2控制：管理通过两个L2高速缓存的流量的控制逻辑。

数据L2：用于除指令以外的所有内存流量的1-MB L2数据缓存。

Instr-L2：1 MB二级指令缓存。

在阅读本书的过程中，本节介绍的这些概念将变得更清晰。

1.3计算机发展简史

在本节中，我们简要概述计算机的发展历程。这段历程本身很有意思，但更重要的是它为本书中的许多重要概念提供了基本介绍。

第一代：真空管

第一代计算机使用数字逻辑元件和存储器的真空管。那时候的许多研究和商业计算机都是用真空管制造的。为了更容易理解，我们来研究最著名的第一代计算机，IAS计算机。

首先在IAS计算机中实现的基本设计方法被称为存储程序概念。这个思想归功于数学家约翰冯诺依曼。 冯诺依曼在1945年的一份新型计算机EDVAC（Electronic Discrete Variable Computer，电子离散变量自动计算机）的计划中首次公布了这一思想。几乎在同时，阿兰图灵也提出了同样的构想。

1946年，冯诺伊曼和他的同事开始在普林斯顿高级研究学院开始设计一种新的储存程序计算机，这种新型机器称为IAS计算机。 虽然直到1952年才完成，但它却成为了后来通用计算机的原型

图1.6给出了lAS计算机的结构（与图1.1进行比较）． 它包括

• 主存储器，用于存储数据和指令。

• 能够处理二进制数的算术逻辑运算单元(ALU) 。

• 控制器， 负责解释内存中指令并执行之。

． 由控制器操纵的输入／输出设备(I/O) 。

冯诺依曼早期的计划中概括了这种结构， 在此值得引用下面的论述（见文献[VONN45]）：

2.2第一 因为这台设备主要见一台计红机， 所以 图2-1 IAS计算机结构 它必须能够执行怂频繁的某本开术运)1:, HP加 ， 减， 乘、 除运算。 因此 ， 它应该包含特殊的器件来执行这些操作。

必须斤到， 虽然这些原理可能是合理的， 但姑需要仔细研究实现它的特殊方式。 无论如何

这一设备的中央打术(eeo,叫,ri,hmdleal)部分必须存在， 它组成了第1个持定的部分 CA。

2.3第二 设备中控制操作顺序的逻辑控制部分， 能够由中央控制器朵打效的实现。 如果此设备有灵活性． 也就是说， 几乎适用于所打应用． 那么必须区分对待给定的指令和所给出的符殊 问题， 以保证这些指令（无论它是什么）都能被通用控制器执行。 前者 必须以某种形式存储． 后者通过定义设备的橾作部分来表不。 中央控制(een<roleoo,rol)仅指后若的功能。 中央控制和 实现它的器件组成了第2个持定部分——CC。

2.4第三任何执行长而复杂的橾作序列（特别是计算序列）的设备都必须有一个相当大的存储器......

(b)管理一个复杂问题的指令集可能包含很多内容， 持别是根据情况而编码（这存在于大多数情况中）， 这些紫材必须被记忆。

无论如何， 所打的存估器(meo邓,1组成了设备的第3个特定的部分

2.6这3个特定的部分. CA 、 cc(一起被称为C)和M, 对应千人类神经系统中的联想神经元。后面还需耍讨论憋觉和运动神经元的对应物， 即设备的输入和输出器件。

设备必须具打接触某些特定媒体并进行输入和输出（感觉和运动）的能力。 这种媒体被称为设备的外部记录媒体：R。

2.7第四 设备必须打从R到特定的C和M传必信息的器件。 这类器件形成了它的输人 (lnp山） ， 因此第 4部分是1。 我们将籵见， 见好使得所有的传送都是从R(通过I)到M, 而绝非直接来自C 。

2.8第上设备必须有从它的特定部分C和M传达信息到R的器件。这些器件形成它的构出 (oo,p山），因此第5部分是0。我们还将斤见，朵好使得所才i的传这都从"I通过O)至R,而绝非直接来自C。

除了少数特例外，当今所有计算机都具有与上述相同的结构和功能，因此它们都被称为冯订伊曼机”。所以值得在这里简单描述一下!AS计算机的操作（参见[BURK46])。依据文献[HAYE98]. 冯·诺伊曼术语和概念变得更贴近当代的用法．伴随这些讨论的示例也基于文献[HAYE98]。

IAS的存储器包含4096个存储单元，它们被称为字（word），每个字有40位（bit）。数据和指令都存储在此。数据被表示为二进制形式，而且指令是二进制编码，图1.7给出了这种形式。每个数被表示为1个符号位和39个数值位。一个字也可以包含两条20位的指令，每条指令包含一个8位的操作码，用来指定所执行的操作类型，和一个12位的地址，用于指定存储器中某个字的地址（0~999）。

控制器通过一次从存储器中取一条指令并执行它的方式来操作IAS。我们参考图1.6来解释这一过程，此图表明无论是控制器还是ALU都包含了存储区域，他们被称为寄存器（register），具体定义如下：

• 存储器缓冲寄存器(MBR),包含将要写到存储器或送到I/O单元或者接受来自存储器或I/O单元的一个字。

• 存储器地址寄存器(MAR),指定将要从 MBR写进存储器或从存储器读入MBR的存储器字单元的地址。

• 指令寄存器(IR),包含正在执行的8位操作码指令。

• 指令缓冲寄存器(IBR)用来暂存来自存储器一个字的右边指令。

• 程序计数器(PC)：存放将要从存储器中获取的下一对指令的地址。

• 累加器(AC)和乘商寄存器(MQ)：用来暂存ALU运算的操作数和结果。例如， 两个40位的数相乘，结果是一个80位的数，则高40位放在AC中，低40位放在MQ中。

IAS通过反复执行指令周期（instruction cycle）来运行，如图1.8所示，每个指令周期由两个子周期组成。在取指周期（fetch cycle）中，下一条指令的操作码装入IR，地址部分装入MAR。指令可以从IBR中获得，也可以从存储器中获得，即先从存储器装载一个字到MBR，然后将该字解开放入IBR、IR和MAR。

为什么不直搂获取？因为这些操作都是巾电子电路控制并且导致数据路径的使用 。 为了简化电路， 只用一个寄存器来 指定存储器中读写的地址， 而且也只用一个寄存器来存放数据源或目标。

一且操作码在IR中， 则进人执行用期(m'"te,yde) 。 控胡电路翻译操作码， 并且通过发送相应的控制佑号来 执行指令 ， 这些信号 控制数据的传送和ALU操作的执行。

IAS计算机共有21条指令 ， 在表1.1中列出 ， 可分为以下几类：

• 数据传送· 在存储器和ALU的寄存器之间或在两个ALU寄存器之间传送数据 。

．无条件转移：通常，控制器按顺序执行存储器中的指令 ， 但这一顺序能通过跳转指令加以改变， 以便执行重复复的操作 。

• 条件转移：可以依据条件来决定是否跳转， 从而选择从何处跳转。

• 算术运算：由ALU完成的操作 。

• 地址修改：允许在ALU中计算地址， 并将它插人存储器的指令中， 为程序寻址带来很大的灵活性。

表1.1以符号化的，易读的形式列出了指令。 事实上， 每条指令必须遵循图2-2b的格式 。橾作码部分（前8位）指定了将要执行的21条指令。地址部分（剩余的12位）指定了执行指令所涉及的4096个存储单元的某一个 。

图1.8给出控制器执行指令的几个例子。注意， 每个操作都要求用几步来完成， 其中有些是相当精巧的。 乘法运算需要39个子操作 ， 除符号位外． 每个位对应一个子操作 。

第二代：晶体管

电子计算机的第一个主要改变是使用晶体管代替电子管。 晶体管比电子管的体积更小、 更便宜，发热更少， 而且能以与电子管相同的方式建造计算机。 电子管由导线、 金属片、 玻璃外壳和其空管构成，而晶体管是固态器件， 由硅片制成。

1947年贝尔实验室发明了晶体管， 从而在20 世纪50年代引发了一场电子革命。直到20世纪50年代末，完全晶体管计算机才能实际应用于商业。晶体管的使用是第二代计算机的标志。 基于所采用的基本硬件技术来划分各个计算机时代， 已成为人们的一个共识（如表2-2所示）。新一代计算机以比旧一代机器具打更快的处理速度、 更大的存储容量和更小的尺寸。

这个时期， 还发生了其他变化。 第二代计算机采用了更复杂的算术逻辑单元和控制器， 使用了高级编程语言，并为计算机提供了系统软件。从广义上说，系统软件提供了加载程序，将数据移到外围设备和库执行常见计算的功能，类似于Windows和Linux等现代操作系统。

研究一下第二代的重要成员：IBM7094[BELL71]，将对我们很有帮助。从1952年出现 700系列到1964年7000 系列的朵后一个产品， IBM的产品经历了计算机产品 的典型演变历程。 每个产品的后续产品都有更奋的性能、 更大的容量和／或者更低的价格。

主存容量是2"个36位字的倍数． 从2K (IK =2")字增加到32K字。访问内存中一个字的时间，称为内存周期，从30μS下降到1.4μS。操作码的数量也从可怜的24种增加到185种。

而且，在该系列计算机的整个使用寿命期间，CPU的相对速度提高了50倍。速度的提高是因为电子元件的改进（例如 ， 品体管比电子管速度更快）和更复杂的电路。 例如 ， IBM 7049包含指令缓存奇存器(IBR), 它用来竹存下一条指令。 控制器在每个指令的取指周期从存储器取相邻的两个字 ， 除非遇到不常发生（ 可能是10到15％）的分支指令， 这意味着控制器只需在一半的指令周期才需要访问内存取指令。 这 一预取指令方式显著减少了平均指令周期时间。

图1.9表示了IBM 7049的大型配置环境（带有许多外设）， 这是第二代计算机的代表。 IBM 7049与IAS计算机的区别值得特别注意，最重要的是它使用了数据通道（data channel）。 数据通道是独立的I/O模块， 具有自己的处理器和指令从。 在带有这些设备的计竹机系统中. CPU不执行具体的VO指令。 这些指令被存放在主存中， 由数据通i1!本身的专用处理器执行 。 CPU通过给数据通迫发送一个控制信号来初始化VO传送， 指示它执行内存中的一中指令。 数 据通过独立地执行它的任务， 并在操作完成时通知CPU。 这种方式减轻了CPU很多的处理负担。

另一个新特点是多路（复用）器（*multiplexor*）, 它是数据通玑 ， CPU, 内存的中心连接贞。 多路器调度CPU和数据通直对内存的访问 ， 允许这些设备独立运行 。

第三代：集成电路

单个独立封装的品体竹称为分立元件(d庄 ,rete oompoaeot) , 从20世纪50年代到60年代早期， 电子设备包含大让的分立元件􀃴晶体 忤 、 电阻、 电容等。 分立元件独立制造 ， 封装在 自己的容器中， 然后一起焊接到或连接到纤维板（类似电路板）上， 垃后再安装到计算机， 示波 器或其他电子设备中。 在电子设备需要品体管的地力 ， 一个包含针尖大小硅片的小品体管就会被焊接到电路板上。 从品体管到电路板的整个制造 过程都是昂贵且麻烦的。

这种情况给计符机工业带来了问题。 早期的第二代计算机包含约JO 000个品体ff. 这一数 字后来增长到数十万， 使得生产更新、 更强大的计算机变得越来越困难。

1958年出现了电子学打i命性的成就， 开创了改电子时代，集成电路的发明。 正怂织成电路定义了第三代计11机。 这一节将简单介绍集成电路技术， 然后 ， 名东可能是第三代计算机中最项要的两个成员 IBM的Sy,1ern/360和DEC的PDP-8, 两者桏在这个时代的初期出现。

微电子技术 顾名思义，微电子技术是“微小的电子技术”。从数字电子技术和计算机工业一开始，就存在持续不断的减小数字电子电路尺寸的趋势。在考察这一趋势的内涵和好处之前，我们需要先介绍数字电子技术的一些特质，而更详细的讨论在第11章。

如我们所知，电子计算机的基本器件必须执行存储，传送，处理，控制等功能。只有两种基本类型的元件是必需的：逻辑门和存储器位元（如图1.10所示）。逻辑门是实现简单布尔或逻辑功能的元件，例如，“如果A和B是真，那么C是真（与门）”。由于他们控制数据流的方式与闸门相似，因此这种设备被称为逻辑门。存储器位元是一个能够存储一位数据的元件，也就是说，该元件在任何时刻都可以处于两个稳定状态之一。将大量的基本元件连接起来，就能够建造一台计算机。我们可以将此与如下4个基本功能联系起来：

数据存储：由存储器位元提供。

数据处理：由逻辑门电路提供。

数据移动：部件间的通路用于将数据从内存传送到内存，或从内存通过门电路再传送到内存。

控制：部件间的通路传送控制信号。例如，一个门有一个或两个数据输入和激活启动门的控制信号输入。当控制信号是ON时，逻辑门对数据输入执行其功能并产生数据输出。相反，当控制信号为OFF时，输出线路为空，例如由高阻抗状态产生的线路。类似地， 存估器位元在写控制信号为ON时， 存人其输入线的位伯，而在读控制信号为ON时，将位元的位置放置在其输出线上。

因此，计算机包含门、 存估器位元和它们之间的互连。 而这些门和存估器位元是由简单的数字电子元件构造的。

从成电路利用了一个邓实， 即晶体管 、 电阻 ， 导线都可以用硅之类的半导体制成。 将整个电路安装在很小的硅片上而不是用分立元件搭成的笱价电路， 只不过是 固态技术” 的一种扩展 ，而且在一块硅品片上能同时制选很多个品体管。 同样政要的是， 这些品体管能够通过金风化过 程相互庄接， 以形成电路。

图1.11描述了集成电路的关键性概念。 一块薄硅品片(wal"')划分由多个小区域排列而成的阵列 、 和个区域有儿平方还米， 它们上面都有相同的电路。 这块晶片被划分成许多块芯片(,hip). 每块芯片都包含许多逻钳门和／或存储器位元以及许多输人、 输出忐接点。 然后封装这块芯片， 使之得以保护， 井加上引脚 ． 用以迕接芯片外部的其他设备。 许多这样的渠成电路块可以迕接在印刷电路板上， 产生更大、 更复杂的电路。

起初 ， 只打儿个门和存估器位元可以可靠地制造井封装在一起。 这些早期的北成电路被称为小规棋集成电路(551) 。 随廿时间的推移， 将越来越多的元件放在同一块芯片上成为可能。 图1.12显示了密度的增长． 它是所打曾经记录的垃显茗技术趋势之一。 此图反映了著名的摩尔定律，该定律是Intel合伙创办人之一高登摩尔(G欧瑞东Moore)于1965年提出的（参见文献【MOOR65】）。摩尔观察到单芯片上所能包含的品体管数从每年翻一番 ， 并正确断官这种态势在不远的将来还会继续下去。 令许多人（包括摩尔在内）惊奇的是， 这种态势年复一年地持续了下来。 直到1970年， 这种态势减慢成每 18个月翻一番， 但从此之后 ， 这个新增长速率又持续下来。

摩尔定律的影响是深远的

1. 在芯片集成度快速增长的期间，单个芯片的成本几乎没有变化。这意味着计算机逻辑电路和存储器电路的成本显著下降。
2. 因为在集成度更高的芯片中逻辑和存储器单元的位置更靠近，电子线路长度更短，所以提高了工作速度。
3. 计算机变得更小，更容易放置在各种环境中。
4. 减小了电能消耗及对冷却的要求。
5. 集成电路内部的连接比焊接更可靠。由于芯片上电路的增加、芯片间的连接变得更少。

IBM System/360

到1964年， IBM的7000系列机已经牢牢占据了市场。 那年. IBM发布了System/360, 这是计算机产品的一个新家族。 虽然发布消息本身井不惊人， 但它包括了一些令现有IBM客户感到 不太舒服的信息： 360产品线与原来的IBM机器不兼容。 因此 ． 现有的客户转移到360将会遇到困难。 这是IBM所迈出的大胆一步 ， IBM认为有必要突破7000体系结构的限制， 制选能随打新 的从成电路技术一起发展的系统[PAOEBI, G!F"7] 。 这一战略在经济上和技术上都会得到回报。 360在那时获得了成功， 巩固了IBM作为计算机销售的绝对优势地位, IBM占有70%以上的 市场份额。 经过相应的修改和扩展， 360体系结构至今仍是IBM大型机的体系结构． 本书将一直采用此结构作为范例。

System/360系列是工业上第一个计算机系列， 该系列机覆盖的性能和价格范围很广。各种型号之间的兼容性体现在， 为系列机中某个型号写的程序可以在系列机中任一个型号上运行，只是运行时间不同。

兼容计算机的系列化概念不但新颖而且取得了成功。 用户起初可以购买较便宜的Model 30 来满足不高的要求和预算。 之后， 如果用户的需要提高了， 可以将它升级到内存更多、 速度更快的机器上， 而不必对已经开发的软件再付出投资。 IBM系列机具有下列特征：

• 相同的或相似的指令集：多数情况下， 系列机中的所有成员都具有完全相同的机器指令

集。这样， 能够在一台机器上执行的桯序同样也能在另一台机器上执行。某些情况下．

系列机中低端产品的指令集是高端产品的一个子集，这意味着程序可以向上移植而不能

向下移植。

• 相似或相同的操作系统：系列机中的所有成员都有相同的基本操作系统．在某些情况下，

高端成员会增添一些新特征。

• 更高的速度：系列机中从低端成员到店端成员， 其指令执行速度逐渐增加。

• 更多的I/O端口数：系列机中从低端成员到环端成员， 其I/O端口数越来越多。

• 更大的内存容量，系列机中从低端成员到高端成员， 其主存容量越来越越大。

• 更高的成本：在某一时间段， 系列机中从低端成员到高端成员． 其成本越来越高。

系列机的概念如何实现？区别在于3个因索：基本速度、容从和井行程度[STEV64] 。例

如， 通过在ALU中使用更复杂的电路， 允许子操作并行地执行， 使钳某一给定指令的执行速度更快。另一个提店速度的方式是增加主存与CPU之间的数据宽度， Modd JO每次只能从主存中读取一个字节(8位） ， 而Model 75每次能够读取8个字节。

Sysleov360不仅指明了IBM未来发展的方向， 而且对整个计疗机行业有抒深远的影响， 其

许多特点已成为其他大型机的标准。

DEC公司的PDP-8

在IBM交付其第一个System/360产品的同一年， 还出现了另一重要的产品------数字装备公

司(DEC)的PDP-8。在多数计算机仍需要空调环境的时候. PDP-8 (在业界称为小型机．在迷

你短裙出现之后）足够小， 以至于可以放在实验室的工作台上， 或被安装在其他系统中。这种小型机不能完成大型机的所有工作， 但价格只要1.6万美元， 这个价格足够便宜， 使得每个实验室技术人员都能配备它。与之相比， 几个月前出现的System/360系列大型机价格高达几十万美元。

PDP-8的低价格、小尺寸使得其他制造商可以购买PDP-8, 并将它集成到自己的计算机系统

中再行出售。这些其他制造商被称为原始设备制造商(OEM), 并且OEM市场成为计算机市场

的主要部分。

与IBM在700/7000和360系列机中所采用的中央交换机结构（如图1.9所示）相比，PDP-8的后续型号采用了一种现在几乎已被所有微型机所采用的结构， 即总线结构 ， 如图1.13所示。 PDP-S的总线被称为Omnibus, 包含96个独立的信号通路 ， 用来传送控制信号 、 地址信号和数据信号。由于所有的系统元件都分享同一套信号通路 ， 因此它们的使用必须由CPU来控制。 这种结构具有高度的灵活性． 允许将模块插入总线以形成各种配置。而在最近几年，总线结构才让位于第3章中描述的称为点对点互连的结构。

后续几代

如何定义计算机第三代以后的各代，意见不太一致，表1.2建议根据集成电路技术的发展来划分以后几代。随着大规模集成电路（LSI）的采用，一块集成电路芯片上可以放置1000个元件，超大规模集成（VLSI）达到每个芯片1万个元件，而现在的巨大规模（ULSI）芯片集成了超过100万个元件。

随着技术的高速发展，产品的迅速更新，软件和通信变得和硬件同等重要。各代之间的界限变得越来越模糊且没有太大意义。本节介绍后续几代中最重要的两次发展。

半导体存储器

集成电路技术在计算机中的 最初应用是采用集成电路芯片来制作处理器（控制单元和算术逻辑单元） ， 但人们同时还发现这一技术也可以构造存储器。

20世纪50年代和60年代， 所有计算机存储器都是由饭小的铁磁体 环做成， 每个立径约1/16 英寸。 这些小 环被吊在计算机内用细线做成的网格上。 一个环（或称为磁芯）的一种磁化方向 代表1, 另一个磁化方向则代表一个0。 磁芯存储器速度相当快， 读存忧器速度中的一位只盆百万分之一秒。 但姑磁芯的价格昂贵， 体积大， 而且读出是破坏性的． 对磁芯的一次读取会擦除其 存估的数据。 因此必须安装读出后立即恢复数据的电路。

1970年，Fairchild (仙童公司）生产了第一个容片较大的半导体存储器。 一块相当于单个磁 芯大小的芯片， 包含了256位的 内存。 这种内存芯片是非破坏性的 ， 而且读写速度比磁芯快很多 读取一位只需要70ns，但其每位的价格比磁芯的要贵。

1974年，一件关键性的事件发生了：半导体存储器的每位价格低于磁芯存储器的每位价格。这以后，存储器的价格持续快速下跌，但物理存储密度不断增加。这导致了新的机器比几年前的机器更小、更快、内存容量更大而且价格更便宜。存储器技术的发展，与将要讨论的处理器技术的发展一起，在不到10年的时间里改变了计算机的生命力。虽然庞大、昂贵的计算机仍然存在，但计算机已经以办公设备和个人电脑的方式走向了“最终用户”。

自从1970年起，半导体存储器经历了13代：单个芯片1k，4k，16k，64k，256k，1M，4M，16M，64M，256M，1G，4G，和现在的8Gb（1 k = 210, 1 M = 220, 1 G = 230）。每代都提供了更高的存储密度，而每位价格和存取时间都在下降。 到2018年密度预计将达到16 Gb，到2023年密度将达到32 Gb [ITRS14]。

微处理器

同 存估器芯片一样 ， 处理器芯片的小元密度也在不断地增加。随籽时间的推移 ， 越来越多的单元被放'l1.在 每＿块芯片上•因此. 构建＿个计)1:机的处理器所需要的心并i召来越少。

1971年出现了突破， '""'升发了'""'4004。 4004是第一 个将CPU的所有元件都放在同一块 芯片内的产品一一于儿 ， 依处理器诞生了。

4004能完成两个4位数相加 ， 通过须复相加来完成乘法。 以今天的标准 ， 4004虽然相对简单，但是它却成为饭处理器的能力和功能不断发展的开端。

这一演变从处理器一次所能处理的位数就可能容易的看出。虽然没有明确的衡量方法，但最好的衡批力法也许就是数据总线的宽度· 处理器能够一次同时输入或输出的数据位数。 另一种衡杖方法是行屎加器或通用寄存器组的数据位数。 虽然打时两种方法得出的结构碰巧相同 ， 但井非总是如此。 例如， 有许多饿处理器的寄存器位数是16位 ， 但其数据总线却只打8位 。

微处理器演变中另一 个主要进步怂1972年出现的'""'8008, 这是第一个8位的改处理器 ， 它儿乎比4004复杂一倍。

这两种产品都 没有下面的这一􀂰·件影响深远 1974年出现了""''8080, 这是第一 个通用饮 处理器。 4004和8008是为特殊Ill迩而设计的 ， 而8080是为通用饮机 设计的中央处理器。 它与 8008一样 ， 都是 8位微处理器 ， 但8080更快 、 有十1:1的指令从且 打更强的寻址能力。

大约在同时， 16位饮处理器被开发出来， 但且到20世纪70年代末才出现强大的通用16位改处理器 ， 8086便怂其中之一。 这一发展凸势中的另一阶段是在1981年 ， 贝尔实验室和HP公司开发出了32位单片饭处理器。 '"叫于1985年推出了其 32位微处理器'""'80386 (如表1.3所示）。

1.4 Intel x86体系结构的进展

贯穿本书，我们依赖计算机设计和实现的许多具体实例来说明各种概念，并阐述各类方法之间的权衡。过去和现在的许多系统都提供了重要的计算机体系结构设计功能的示例，本书主要依据两个处理器系列的例子：Intel x86和ARM架构。当代的Intel x86代表了在复杂指令集计算机(CISC)中几十年设计成果的结晶，它采用了过去只有大型机和超级计算机中才会采用的复杂设计原则， 是CISC设计的优秀范例。 处理器设计的另一种方法是稍简指令集计算机(RISC)。ARM体系结构被广泛应用于各种类型的嵌入式系统，它是市场上基于RISC技术的功能最强大、设计最好的系列之一。下面将简要介绍这两个系统。

依据市场份额，Intel公司在过去的几十年中始终是微处理器（非嵌入式系统）的领先制造商，这一地位似乎难以动摇。其旗舰的微处理器产品的演变历史，是整个计算机技术演变的一个很好的写照。

表1.3给出了这一演变历史。有趣的是，当微处理器变得更快更复杂时，Intel的步调总是那么合拍。过去，Intel习惯于每4年开发出一种新的处理器，但它期望将每代开发时间缩短到1-2年，以继续保持领先优势，并且已经这样做了，从而加速推进了几代最新的X86产品。

列出Intel产品系列的一些主要进展可以帮助我们理解：

• 8080:世界上第一台通用微处理器。 它是8位机， 存储器的数据通路为8位。 8080曾用于第一台个人计算机Altair。

• 8086:比8080 更强大的16位微处理器。 除了更宽的数据通路和更大的寄存器外， 8086还支持指令高速缓存（或称为队列）。在指令被实际执行之前，它能预取几条指令。 这种处理器的一个变形是8088, 8088曾用于IBM公司的第一台个人计算机， 并确保了Intel的成功。 8086标志着x86体系结构的首次出现。

• 80286：它是 8086的扩展产品 ， 可以寻址16MB的存储器， 而不是1MB存储空间。

• 80386：Intel的第一个32位机器， 是一个有重大改进的产品。 其32位的体系结构， 使80386的复杂程度和功能可以与几年前推出的小型机和大型机相媲美。80386是Intel公司第一个支持多任务的处理器， 即它能够同时运行多个程序。

• 80486：80486采用了更为复杂、 功能更强的高速缓存技术和指令流水线技术。 它的内置式的数学协处理器， 减轻了主处理器的复杂算术运算的负担。

• Pentium：从Pentium开始，Intel推出了超标量技术， 它允许多条指令并行地执行 。

• Pentium Pro：Pentium Pro继续推进由Pentium开始的超标量结构， 极富进取性地使用了包括寄存器重命名 、 分支预测 、 数据流分析、 推断执行等技术。

• Pentium Ⅱ:融入了专门用于有效处理视频、 音频和图形数据的lntel MMX技术。

• Pentium III：Pentium III集成了更多的浮点指令，流式SIMD扩展（SSE）指令集增加了70条新指令，旨在提高在多个数据对象上执行完全相同的操作时的性能。 典型的应用是数字信号处理和图形处理。

• Pentium 4, 包括了另一些浮点指令， 并对其他多媒体应用进行了增强

• Core, 这是第一款具有双核的lntel x86处理器， 涉及在单芯片上双处理器的实现。

• Core 2：Core 2将体系结构扩展到64位, Core 2 Quad在单芯片上提供了4个内核。最新的Core每个芯片最多有10个内核。 该架构的一个重要更新是高级矢量扩展指令集，该指令集提供了一组用于有效处理矢量数据的一组256位和512位指令。

Intel x86体系结构自1978年推出至今已有40多年， 它一直统治着除嵌人式系统之外的处理器市场。 虽然x86机器的组成和技术在几十年间发生着戏剧性的变化，但其指令集结构一且保持着向后兼容其早期的版本， 因此 ，任何写于x86体系结构早期版本的程序都可以在其更新的版本上运行 。 所有对指令集结构的改变都是添加指令集， 而不是减少指令集。 指令集的变化速度大约是每月增加1条其他指令[ ANTH08], 因此， 目前x86指令集有上千条指令。

Intel x86极好地说明了过去35年来计算机硬件的发展。 1978年推出的8086, 其时钟频率为5MHz, 包含29000个晶体管。 而2 013年推出的 6核Core i7 EE 4960X, 其主频为4GHz, 是8086的800 倍，它包含18.6亿个晶体管． 大约足8086的64 000倍。 然而 ， Core i7 EE 4960X只是比8086的封装稍微大一点，且价格可比。

1.5嵌入式系统

术语“嵌入式系统”是指电子学和软件在产品中的使用，它与通用计算机系统不同，例如笔记本电脑或台式机。每年会出售数百万台计算机，包括笔记本电脑，个人电脑，工作站，服务器，大型机和超级计算机。相比之下，每年都会生产数十亿计算机系统，这些计算机系统嵌入在更大型的设备中。如今许多使用电力的设备或许大部分都具有嵌入式计算系统。很可能在不久的将来，几乎所有这些设备都将具有嵌入式计算系统。

具有嵌入式系统的设备类型几乎无法列举。例如手机，数码相机，摄像机，计算器，微波炉，家庭安全系统，洗衣机，照明系统，恒温器，打印机，各种汽车系统（例如变速箱控制系统，巡航控制系统，燃油喷射系统，防抱死制动系统和悬挂系统），网球拍，牙刷以及自动化系统中众多类型的传感器和执行器。

通常 ， 嵌入式系统与其环境密切相关。 这能够通过环境的相互影响而引起实时限制。 所有限制、例如运动速度的要求、 测社精度的要求和时间期间的要求． 表明了软件操作的定时。 如果必须同时处理多个行为，则包含更复杂的限制。

图1.14给出了一个嵌入式系统组成的基本术语。 除了处理器和存储器，还有很多不同于典型台式机或笔记本电脑的元件：

可能有各种用于测量、操作和其他与外部环境交互的界面。 嵌入式系统通常通过传感器和执行器与外部世界相互作用（感知，操作和交流），因此通常是反应式系统; 反应系统与环境持续地相互作用，并且以由该环境决定的速率执行。

人机界面可以像闪光灯一样简单，也可以像实时机器人视觉那样复杂。 在很多情况下，没有人机界面。

诊断端口可用于诊断正在控制的系统 - 不仅用于诊断计算机。

特殊用途的现场可编程门阵列(FPGA) 、专用集成电路（ASIC），甚至非数字式硬件，可以用来增强性能或可靠性

软件通常具有固定的功能并且对该应用有特效

效率对嵌入式系统至关重要。 它们针对能源，代码长度，执行时间，重量和尺寸以及成本进行了优化。

通用计算机系统还有几个值得注意的地方：

即使使用固定功能的软件，即时升级修复错误的能力，提高安全性和增加功能的能力，不仅对用户设备很重要，对于嵌入式系统而言也非常重要，

嵌入式系统平台的一个相对较新的发展是支持各种各样的应用程序。 这方面比较好的例子是智能手机和音频/视频设备，例如智能电视。

物联网

区分出嵌入式系统普及的主要推动力之一是值得做的。“物联网（IoT）”是一个术语，指的是智能设备之间的互联互通，从家电到微型传感器。一个主要的方法是将短程移动收发器嵌入各种小工具和日常物品中，实现人与物之间以及物件之间的新型通信。互联网现在支持数十亿个工业和个人对象的互联，通常是通过云系统。这些对象提供传感器信息，根据其环境进行操作，并且在某些情况下可以自行修改，以创建更大系统的整体管理，如工厂或城市。

物联网主要由深度嵌入式设备（定义如下）驱动。这些设备是低带宽，低重复率数据捕获和低带宽数据使用设备，可相互通信并通过用户界面提供数据。

嵌入式设备（如高分辨率视频安全摄像头，视频VoIP电话和其他少数设备）需要高带宽流媒体功能。然而，许多产品只需要间歇性地提供数据包。

关于支持的终端系统，互联网已经历了大约四代的部署，最终形成了物联网：

1.信息技术（IT）：购买PC，服务器，路由器，防火墙等作为企业IT人员的IT设备，主要使用有线连接。

2.操作技术（OT）：非互联网公司的内置的IT机器/设备，如医疗机械，SCADA（监督控制

和数据采集），过程控制和信息亭，企业OT人员购买，主要使用有线连接。

3.个人技术：消费者或员工购买智能手机，平板电脑和电子书阅读器作为IT设备，只使用无线连接并且通常是多种形式的无线连接。

4.传感器/执行器技术：消费者，IT人员和OT人员购买的有特定用处设备，使用无线连接并且通常是大型系统的一部分。

IoT通常被认为是物联网的第四代产品，它的特点是使用数十亿台嵌入式设备。

嵌入式操作系统

开发嵌入式操作系统（OS）一般有两种方法。第一种方法是使现有的操作系统适用于嵌入式应用程序。例如，嵌入式版本的Linux，Windows和Mac以及专用于嵌入式系统的其他商用和专有操作系统。另一种方法是设计和实现专为嵌入式应用而设计的操作系统。后者的一个例子是TinyOS，它广泛用于无线传感器网络。该问题在[STAL15]中进行了深入探讨。

应用处理器与专用处理器

下面我们简要介绍有关嵌入式系统的文献中常见的一些术语。应用处理器由处理器执行复杂操作系统（如Linux，Android和Chrome）的能力来定义。因此，应用程序处理器本质上是通用的。嵌入式应用处理器的一个很好的例子就是智能手机。设计嵌入式系统用于支持众多的应用程序并执行多种功能。

大多数嵌入式系统采用专用处理器，顾名思义，专用处理器专用于主机设备所需的一项或少量特定任务。由于这种嵌入式系统专用于某种或某些特定任务，因此可以设计专门的处理器和相关组件以减小尺寸和成本。

微处理器与微控制器

可以看到，早期的微处理器芯片包括寄存器，ALU以及某种控制单元或指令处理逻辑。 随着晶体管密度的增加，增加指令集架构的复杂性，最终增加内存和多个处理器也成为可能。 如图1.2所示，当代微处理器芯片包括多个内核和大量的高速缓存。

微控制器芯片对可用的逻辑空间的使用大不相同。 图1.15表示了微控制器芯片上常见的元件。 如图所示，微控制器是单个芯片，它包含处理器，用于程序的非易失性存储器（ROM），用于输入和输出（RAM）的易失性存储器，时钟和I / O控制单元。 微控制器的处理器部分具有比其他微处理器小得多的硅面积以及更高的能效。 我们在1.6节更详细地研究微控制器结构。

微控制器也被称为单片计算机，每年都有数十亿个微控制器单元嵌入从玩具到电器到汽车的各种产品中。 例如，一辆车可以使用70个或更多的微控制器。 通常，微控制器（特别是小而便宜的）会被用作特定任务的专用处理器。 例如，微控制器在自动化过程中大量使用。 通过提供简单的输入反应，他们可以控制机器，打开和关闭风扇，打开和关闭阀门等等。 它们是现代工业技术不可或缺的组成部分，也是生产可以处理极其复杂功能的机器的最划算的途径之一。

微控制器有许多物理尺寸和不同的处理能力。 处理器的范围从4位到32位架构。 微控制器往往比微处理器慢得多，通常在MHz范围内运行，而不是微处理器的GHz速度。 微控制器的另一个典型特征是它不提供人机交互。 微控制器针对特定任务进行编程，嵌入在设备中，并在需要时执行。

嵌入式与深度嵌入式系统

本节我们定义了嵌入式系统的概念。嵌入式系统相当多的子集被称为深度嵌入式系统。虽然这个术语在技术上和商业文献中被广泛使用，但当你搜索互联网上的直接定义时将会一无所获（或者至少我是这样的）。一般而言，我们认为深度嵌入式系统有一个处理器，其行为难以被程序员和用户观察到。深度嵌入式系统使用微控制器而不是微处理器，一旦设备的程序逻辑被烧入ROM（只读存储器）中便不可编程，并且与用户没有交互。

深度嵌入式系统是专用的单一设备，用于检测环境中的某些内容，执行基本级别的处理，然后对结果进行处理。深度嵌入式系统通常具有无线功能并出现在网络化配置中，例如大面积部署的传感器网络（例如工厂，农业领域）。物联网在很大程度上依赖于深度嵌入式系统。通常情况下，深度嵌入式系统在内存，处理器尺寸，时间和功耗方面存在极其严格的资源限制。

1.6ARM体系结构

ARM体系结构是指其处理器结构遵循RISC设计原则，并用于嵌入式系统之中。第15章将详细说明RISC的设计原则。本节将简要介绍ARM体系结构。

ARM的进展

ARM是一种由英国剑桥ARM公司设计的基于RISC的微处理器和微控制器序列。 该公司井不生产处理器， 而是设计微处理器和多核的体系结构， 然后向制造商发放许可。具体而言，ARM控股有两种类型的许可产品：处理器和处理器架构。 对于处理器，客户购买使用ARM提供的在他们自己芯片上进行设计的权利。 对于处理器架构，客户购买设计符合ARM架构的处理器的权利。

ARM芯片是高速的处理器， 这是因为它们的小特征尺寸和低能耗需求。它们被广泛应用于PDA 以及其他手提设备中 ， 包括游戏机以及各种消费产品。 ARM芯片是苹果公司流行的iPod和iPhone设备的处理器，并且几乎可用于所有Android智能手机。 ARM可能是最广泛使用的嵌人式处理器体系结构 ， 并且确实是世界上各种应用中使用最广泛的处理器体系结构【VANC14】。

ARM技术的起税可以追溯到英国的Acom计杆机公司。 在20世纪80年代早期 ， Acom获得了英国广播公司( BBC)的合同 ， 为其公司的计算机文化项目开发一款新的微计算机体系结构。这个合同的成功促使Acom继续开发出其第一款商用RISC商业处理器, ARM (Acom RISC Machine)。 第一个版本ARM1, 在1985年变成可用 ， 并用于内部的研究和开发， 以及用于BBC机 器上作为协处理器。

在这个早期阶段， Acom利用公司的VLSI 技术做处理器芯片的实际制造。 VLSI允许授权其自己的芯片市场， 同时使得其他一些公司在其产品中使用ARM也获得了一些成功， 尤其是在嵌入式处理器中。

ARM设计顺应了嵌入式应用中对高性能、 低功耗、 小体积和低成本的处理器的不断增长的商业需求。 但进一步发展超出了Acom的能力范围 ， 于是 ， 由Acom、 VLSI以及苹果计算机作为股东， 成立了一家新公司， 叫ARM Ltd。 Acom的RISC机器变成了先进的RISC计算机。

指令集架构

ARM指令集非常规则，旨在有效实现处理器和高效执行。 所有指令都是32位长，并遵循常规格式。 这使得ARM ISA适用于多种产品。

增强基本的ARM ISA是Thumb指令集，它是ARM指令集的重新编码子集。 Thumb旨在提高使用16位或更窄内存数据总线的ARM实现的性能，并允许比ARM指令集提供更好的代码密度。 Thumb指令集包含重新编码为16位指令的ARM 32位指令集的子集。 目前定义的版本是Thumb-2。

ARM和Thumb-2 ISA在第12章和第13章中讨论。

ARM产品

ARM Holdings授权许多专用微处理器和相关技术，但其大部分产品线都是Cortex系列微处理器架构。 有三种Cortex体系结构，简要标记为首字母A，R和M.

Cortex-A/Cortex-A50 Cortex-A和Cortex-A50是面向智能手机和电子书阅读器等移动设备以及数字电视和家庭网关（如DSL和有线互联网调制解调器）等消费类设备的应用处理器。 这些处理器以更高的时钟频率（超过1 GHz）运行，并支持内存管理单元（MMU），这是全功能操作系统（如Linux，Android，MS Windows和移动操作系统）所必需的。 MMU是通过将虚拟地址转换为物理地址来支持虚拟内存和寻呼的硬件模块; 这个问题将在第8章中探讨。

这两种架构使用ARM和Thumb-2指令集; 主要区别在于Cortex-A是32位机器，而Cortex-A50是64位机器。

Cortex-R Cortex-R旨在支持实时应用，其中事件的时序需要通过对事件的快速响应来控制。它们可以以相当高的时钟频率（例如，200MHz至800MHz）运行并具有非常低的响应延迟。 Cortex-R包括对指令集和处理器组织的增强，以支持深度嵌入式实时设备。这些处理器中的大多数没有MMU;有限的数据需求和有限数量的同步进程消除了对虚拟内存的复杂硬件和软件支持的需求。 Cortex-R具有存储器保护单元（MPU），高速缓存和为工业应用设计的其他存储器功能。 MPU是一个硬件模块，它禁止内存中的一个程序意外地访问分配给另一个活动程序的内存。使用各种方法，在程序周围创建一个保护边界，禁止程序内的指令引用该边界之外的数据。

使用Cortex-R的嵌入式系统的例子有汽车制动系统，大容量存储控制器以及网络和打印设备。

Cortex-M Cortex-M系列处理器主要针对微控制器领域开发，其中需要快速、高确定性的中断管理与极低的门数和最低可能的功耗的需求相耦合。 与Cortex-R系列一样，Cortex-M架构也有一个MPU，但没有MMU。 Cortex-M仅使用Thumb-2指令集。 Cortex-M的市场包括物联网设备，用于工厂和其他企业的无线传感器/执行器网络，汽车车身电子设备等。

目前有四种版本的Cortex-M系列：

Cortex-M0：专为8位和16位应用而设计，该型号强调低成本，超低功耗和简单性。 它针对小型硅芯片尺寸（从12k门开始）进行了优化，并用于成本最低的芯片中。

Cortex-M0 +：更加节能的M0增强版。

Cortex-M3：专为16位和32位应用而设计，该型号强调性能和能效。 它还具有全面的调试和跟踪功能，使软件开发人员能够快速开发他们的应用程序。

Cortex-M4：该型号提供了Cortex-M3的所有功能，并附带说明以支持数字信号处理任务。

在本文中，我们主要使用ARM Cortex-M3作为示例的嵌入式系统处理器。 它是所有适用于通用微控制器的ARM模型的最佳选择。 Cortex-M3被各种微控制器产品制造商使用。 来自主要合作伙伴的微控制器设备已经将CORTEX-M3处理器与Flash、SRAM和多个外围设备相结合，以1美元的价格提供有竞争力的产品。

图1.16提供了Silicon Labs的EFM32微控制器的框图。该图还显示了Cortex-M3处理器和核心组件的细节。我们依次研究每个层级。

Cortex-M3内核的指令和数据使用不同的总线。这种方式有时被称为哈佛架构，与冯诺依曼架构相反，冯诺依曼架构为指令和数据使用相同的信号总线和存储器。因为能够同时读取内存中的指令和数据，Cortex-M3处理器可以并行执行许多操作，从而加速应用程序的执行。该内核包含一个用于Thumb指令的解码器，一个支持硬件乘法和除法的高级ALU，控制逻辑以及与处理器其他组件的接口。特别是嵌入式向量中断控制器（NVIC）和嵌入式跟踪宏单元（ETM）模块有一个接口。

内心是一个模块的一部分，称为CordX-M3处理器。这个术语有点误导，因为通常在文献中，术语内核和处理器被视为等同。除了内核之外，处理器还包括以下元素：

NVIC：为处理器提供可配置的中断处理能力。它有助于低延迟异常和中断处理，并控制电源管理。

ETM：一个允许调试程序执行的可选调试组件。 ETM被设计为一种仅支持指令跟踪的高速低功耗调试工具。

调试访问端口（DAP）：提供了一个接口，用于对处理器进行外部调试访问。

调试逻辑：基本调试功能包括处理器暂停、单步执行、处理器核心寄存器访问、无限软件断点和全系统内存访问。

ICode接口：从代码存储空间获取指令。

SRAM和外围接口：读/写数据存储器和外围设备接口。

总线矩阵：将内核和调试接口连接到单片机上的外部总线。

内存保护单元：保护操作系统用户应用程序使用的关键数据，通过禁止访问彼此的数据来分离处理任务，禁止访问内存区域，允许将内存区域定义为只读内存区域，仅检测可能会破坏系统的意外内存访问。

图1.16的上半部分说明了使用Cortex-M3构建的典型微控制器的框图，在这种情况下是EFM32微控制器。该微控制器在市场上广泛用于各种设备，包括能源，燃气和水表计量;警报和安全系统;工业自动化设备;家庭自动化设备;智能配件;和健康和健身设备。硅芯片由10个主要区域组成：

内核和内存：该区域包括Cortex-M3处理器，静态RAM（SRAM）数据存储器，和用于存储程序指令和非变化应用程序数据的闪存。 闪存是非易失性的（当电源关闭时数据不会丢失），因此是理想的。 SRAM存储可变数据。 该区域还包括调试接口，可以很容易重新编程和更新系统。

并行I/O端口：可配置多种并行I/O方案。

串行接口：支持各种串行I / O方案。

模拟接口：模拟-数字和数字-模拟逻辑，以支持传感器和执行器。

定时器和触发器：跟踪定时和计数事件，输出波形，并触发其他外设的定时操作。

时钟管理：控制芯片上的时钟和振荡器。 多个时钟和振荡器用于最小化功耗并缩短启动时间。

能量管理：管理处理器和外围设备的各种低能耗模式，以提供能量需求的实时管理，以使能量消耗最小化。

安全性：芯片包含高级加密标准（AES）的硬件实现。

32位总线：连接芯片上的所有组件。

外设总线：允许不同外设模块直接相互通信而不涉及处理器的网络。 支持定时的关键操作并减少软件开销。

比较图1.16和图1.2，可以看到很多相似之处和大致相同的层次结构。 但是要注意微控制器计算机系统的顶层是单芯片，而对于多核计算机，顶层是包含多个芯片的主板。 另一个值得注意的区别是，Cortex-M3处理器和整个微控制器都没有缓存，如果代码或数据在外存储器中，则缓存起着重要作用。 虽然读取指令或数据的周期数取决于缓存命中或未命中，但使用外存储器时，缓存大大提高了性能。 这样的开销对于微控制器是不需要的。

1.7云计算

虽然云计算的一般概念可追溯到20世纪50年代，但云计算服务在21世纪初才首次推出，特别针对大型企业。 从那时起，云计算已经扩展到中小型企业，最近又扩展到消费者。 苹果的iCloud于2012年推出，在推出一周内就有2000万用户。 Evernote是基于云计算的笔记和存档服务，于2008年推出，在不到6年的时间里接近1亿用户。 在本节中，我们做了简要的概述。 云计算在第17章中有更详细的介绍。

基本概念

许多组织一个日益突出的趋势是将相当一部分甚至所有的信息技术（IT）操作转移到被称为企业云计算的互联网连接基础设施上。与此同时，个人电脑和移动设备的用户越来越多地依赖于云计算服务来使用个人云计算来备份数据、同步设备和共享。NIST在NIST SP-800—145（云计算的NIST定义）中定义云计算，如下：

云计算：一种可以随时随地访问共享的可配置计算资源池（例如网络，服务器，存储，应用程序和服务）的模型，可以用最少的管理工作快速配置和发布，或者服务于提供商交互。

基本上，通过云计算可以获得规模经济，专业网络管理和专业安全管理。 这些功能可能对大小公司，政府机构以及个人PC和移动用户都具有吸引力。 个人或公司只需支付他们所需的存储容量和服务的费用。 无论是公司还是个人，用户都无需设置数据库系统，获取所需的硬件，进行维护以及备份数据，这些都是云服务的一部分。

理论上，使用云计算来存储数据并与其他人共享数据的另一大优势是云提供商负责安全管理。 然而客户并不总是受到保护。 云提供商有时会出现一些安全故障。 Evernote在2013年初告诉所有用户因发现入侵需要重置密码时成为了头条新闻。

云网络是指为实现云计算而必须启用的网络和网络管理功能。 大多数云计算解决方案都依赖于互联网，但这只是网络基础设施的一部分。云网络的一个例子是提供商和用户之间提供高性能和/或高可靠性网络。 在这种情况下，企业与云之间的部分或全部流量绕过互联网，并使用云服务提供商拥有或租用的专用网络设施。更一般地说，云网络指的是访问云所需的网络能力的集合，包括利用互联网上的专门服务，将企业数据中心链接到云，并在关键点使用防火墙和其他网络安全设备来执行访问安全策略。

我们可以将云存储视为云计算的一个子集。 实质上，云存储由远程托管在云服务器上的数据库存储和数据库应用程序组成。 云存储使小型企业和个人用户能够利用随需求扩展的数据存储，并免费使用各种数据库应用程序，维护和管理存储资产。

云服务

云计算的基本目的是提供便利的计算资源租用。 云服务提供商（CSP）维护通过Internet或专用网络的计算和数据存储资源。客户可以根据需要租用这些资源的一部分。 几乎所有的云服务都是使用下面三种模式之一（图1.17）：SaaS，PaaS和IaaS，本节我们将会讨论。

软件即服务（SaaS）顾名思义，SaaS云以软件形式向客户提供服务，特别是应用程序软件，在云上运行并可在云中访问。 SaaS遵循熟悉的Web服务模型，在这种情况下应用于云资源。 SaaS使客户能够使用云提供商的应用程序并在供应商的云基础设施上运行。 应用程序可通过简单的界面（如Web浏览器）从不同客户端设备访问。 企业无需为其使用的软件产品获取桌面和服务器许可证，而是从云服务获得相同的功能。 SaaS节省了软件安装，维护，升级和补丁的复杂性。 此级别的服务示例包括Gmail，Google的电子邮件服务和Salesforce.com，这帮助企业追踪他们的用户。

SaaS的普通用户是一些希望为其员工提供典型办公生产力软件（例如文档管理和电子邮件）的组织。 个人也通常使用SaaS模型来获取云资源。 通常，用户按需使用特定的应用程序。 云提供商一般还提供与数据相关的功能，例如用户之间的自动备份和数据共享。

平台即服务（PaaS）PaaS云以客户应用程序可运行平台的形式向客户提供服务。 PaaS使客户能够部署到包含客户创建或获取应用程序的云基础架构上。 PaaS云提供有用的软件构建模块，以及许多开发工具，如编程语言，运行时环境和其他帮助部署新应用程序的工具。 实际上，PaaS是云中的一个操作系统。 对于希望开发新的或定制应用程序的组织而言，PaaS非常有用，同时仅在需要时才支付所需的计算资源，Google App Engine和Salesforce.com的Salesforce1 Platform都是PaaS的示例。

基础架构即服务（IaaS）通过IaaS，客户可以访问底层云基础架构。 IaaS提供虚拟机和其他抽象的硬件和操作系统，可以通过服务应用程序编程接口（API）进行控制。 IaaS提供客户处理，存储，网络和其他基础计算资源，以便客户能够部署和运行任意软件，其中可以包括操作系统和应用程序。 IaaS使客户能够将基本计算服务（如数字处理和数据存储）结合起来，构建高度适应性的计算机系统。 IaaS的例子是Amazon Elastic Compute Cloud（亚马逊EC2）和Windows Azure。

1.8 关键词、思考题和习题

关键词

application processor： 应用处理器

arithmetic and logic unit(ALU):算数逻辑单元

ARM：高级精简指令集计算机

central processing unit(CPU)：中央处理器

Chip：芯片

cloud computing：云计算

cloud networking：云网络

cloud storage：云存储

computer architecture：计算机体系结构

computer organization：计算机组成

control unit：控制单元

Core：内核

dedicated processor：专用处理器

deeply embedded system：深度嵌入式系统

embedded system：嵌入式系统

gate：门

infrastructure as a service(IaaS)：基础架构即服务

input–output(I/O)：输入输出

instruction set architecture(ISA)：指令集架构

integrated circuit：集成电路

Intel x86：英特尔x86

Internet of things (IoT)：物联网

main memory：主存

memory cell：存储单元

memory management unit(MMU)：内存管理单元

memory protection unit(MPU)：存储器保护单元

microcontroller：微控制器

Microelectronics:微电子

Microprocessor：微处理器

Motherboard：主板

Multicore：多核

multicore processor：多核处理器

original equipment：原始设备

manufacturer (OEM)：制造商

platform as a service(PaaS)：平台即服务

printed circuit board：印刷电路板

Processor：处理器

Registers：寄存器

Semiconductor：半导体

semiconductor memory：半导体内存

software as a service (SaaS)：软件即服务

system bus：系统总线

system interconnection：系统互连

vacuum tubes：真空管

思考题

1.1计算机组成与计算机体系结构在概念上有何区别？

1.2 中央处理器与内核在概念上有何区别？

1.3 计算机的四个主要功能是什么？

1.4 列出并概要定义计算机的主要结构部件。

1.5 列出井概要定义处理器的主要结构部件。

1.6多路复用器的意义是什么？

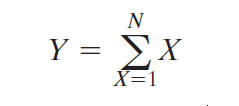
1.7解释摩尔定律

1.8什么是深度嵌入式系统？

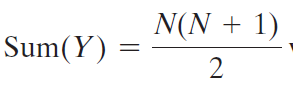
1.9列出并简要讨论Cortex-M系列的四个版本。

习题

1.1编写一个IAS程序来计算以下等式的结果。



假定不会算术溢出，并且X，Y和N是正整数且N≥1。注意：IAS没有汇编语言，只有机器语言。

1. 使用等式编程
2. 用其他方法实现

1.2（1）在IAS上，机器码指令如何将累加器的内容存储到存储器地址8？

（2）在IAS上，机器码指令如何将存储器地址16的内容添加到累加器？

1.3 IAS通过重复执行一个指令周期来运行，该周期由两个子周期组成：一个取指周期和一个执行周期。描述一下IAS在取指周期内完成的任务和在执行周期内完成的任务。

1.4 根据如下所示的IAS计算机的内存内容，

地址 内容

08A 010FA210FB

08B 010FA0F08D

08C 020FA210FB

列出程序的汇编语言代码，从地址08A开始。 说明这个程序的功能。

1.5 在图1.8所示的取指周期中，为什么总是从IBR中获取指令？

1.6 在IBM 360型号65和75中，地址在两个单独的主存储器单元中交错排列（例如，一个单元中的所有偶数字和另一个单元中的所有奇数字）。说明为什么采用这种方法？

1.7 IBM 360型号75的相对性能是360 型号30的50倍，但指令周期只有5倍。 如何解释这种差异？

1.8 你在比利鲍勃的电脑商店闲逛时无意中听到一位顾客向比利鲍勃询问他商店中最快的电脑是什么。 比利鲍勃回复说：“你正在看我们的Macintoshes。 我们速度最快的Mac以1.2 GHz的时钟速度运行。如果您真的想要最快的机器，则应该购买我们的2.4 GHz Intel Pentium IV。“比利鲍勃是否正确？ 你会怎样来帮助这个客户？

1.9 ENIAC是ISA机器的前身，是一台十进制机器，其中每个寄存器由10个真空管组成。 在任何时候，只有一个真空管处于ON状态，代表10个十进制数字中的一个。 假设ENIAC能够同时使多个真空管处于ON和OFF状态，说明为什么这种表示“浪费”，我们可以使用10个真空管表示什么范围的整数值？

1.10 对以下每项，说明是否是嵌入式系统，并解释为什么。

A.理解物理或硬件的实现是嵌入式的吗？例如，使用有限元方法来预测飞机机翼上的流体流动？

B.内部微处理器控制磁盘驱动器是嵌入式系统的一个例子吗？

C.I / O驱动程序控制硬件，因此I / O驱动程序的存在意味着执行驱动程序的计算机是嵌入式的吗？

D.PDA是嵌入式系统吗？

E.微处理器是否使手机成为嵌入式系统？

F.大型相控阵雷达中的电脑是否被认为是嵌入式的？这些雷达是10层高的建筑物，在建筑物的斜坡上有一至三个直径为100英尺的散热片。

G.传统的飞行管理系统（FMS）嵌入到飞机驾驶舱内是否被认为是嵌入式系统？

H.硬件在环（HIL）模拟器中的计算机是嵌入式吗？

I.计算机控制人胸部起搏器是嵌入式系统吗？

J.控制汽车发动机中控制燃油喷射的计算机是嵌入式吗？

第二章

本章讨论计算机系统的性能问题。 我们首先考虑对计算机资源平衡使用的需要，这为整本书提供了有用的观点。下面我们来看一下为满足当前和预计的需求旨在提供良好性能的当代计算机组织设计。 最后，我们看看已经开发的工具和模型，以提供评估比较计算机系统性能的方法。

2.1性能设计

计算机系统的价格都在逐年下降，而他们的性能和容量却在显著提高。现在的笔记本电脑具有与10到15年前IBM大型机相同的计算能力。因此，我们拥有几乎是“免费”的计算机功能。处理器非常便宜以至于我们现在有了微处理器。数字多产测试就是一个例子（使用一次然后扔掉）。而这种持续的技术革命使开发极复杂和极高性能的应用成为可能。 例如，今天基于微处理器系统的功能强大的桌面应用包括：图像处理、三维渲染、语音识别、视频会议、多媒体创作、文件的语音和视频注解、模拟建模。

工作站系统现在支持高度复杂的工程和科学应用，并且有能力支持图像和视频应用程序。 此外，企业正依靠日益强大的服务器来完成交易和进行数据库处理，并支持大型客户端/服务器网络，这些网络已经取代了昔日的大型主机计算机中心。 同样，云服务提供商使用大量高性能服务器库来满足广泛的客户端高容量，高交易率的应用。

从计算机组成和结构的角度来看, 发人深省的是：一方而 ， 组成今天计算机奇迹的基本模块与50年前的IAS计算机基本相同； 另一方面 ， 从现有材料中挤出最后一丁点性能的技术都变得日益复杂。

这一观察结果是陈述本书的指导原则。当考察计算机各个组成部件的时候，我们追求两个目标：第一，本书解释每个所考察领域的基本功能；第二，本书探寻为达到最大性能所要求的技术。本节的剩余部分将突出性能设计所涉及的关键因素。

微处理器的速度

Intel x86处理器或IBM大型机如此震撼人心的强大功能来自处理器芯片制造商对速度的执着追求。 这些机器的演变一直遵循前面介绍的摩尔定律。 只要这个定律保持有效， 芯片制造商就能 每3年发布新一代的芯片， 其晶体管数为上一代芯片的4倍。 对于内存芯片， 仍旧采用基本的主存储器技术 ， 动态随机存储器(DRAM)的容量每3年提高4倍。 对于微处理器， 通过增加新的电路， 减小电路间的距离来提高速度， 使得性能每3年提高4-5倍 ， 从1987年开始推出的Intel x86系列也是如此。

但是， 除非以计算机指令的形式不断向它提供持续的工作流， 否则微处理器将达不到它的 潜在速度。 任何阻碍工作流的事件都会降低处理器的功能。 因此， 当芯片制造商忙于研究怎样不断提高芯片集成度的同时， 处理器的设计者必须提供更加复杂的技术来填饱这个怪物。 当代处理器所包含的技术有：

流水线：指令的执行涉及多个操作阶段，包括获取指令，解码操作码，获取操作数，执行计算等。 流水线使处理器能够同时执行多个指令，同时对多个指令执行不同的阶段。 处理器通过将数据或指令移动到管道中来进行重叠操作，并同时处理管道的每个阶段。 例如，当一条指令正在执行时，计算机正在解码下一条指令。 这与装配线中的原理相同。

转移预测：处理器提前考察取自内存的指令代码， 并预测哪条分支指令或哪组指令可能 下一步将会被执行。 如果处理器大部分时间的猜测是正确的， 则它能预取正确的指令， 并将它们放入缓存， 这样处理器就会始终处于繁忙之中。 这种预测策略的更复杂例子是不只预测下面一个分支， 还要提前预测多条分支。 如此， 转移预测可能增加了可供处理器执行的工作量。

超标量执行：这是每个处理器时钟周期内发出多个指令的能力。 实际上使用了多条平行管线。

数据流分析：处理器通过分析哪一条指令依赖其他指令的结果或数据， 来优化指令调度。 事实上， 准备好的指令就可以被调度执行， 不必按照原来程序的顺序， 这减少了不必要 的延时。

推测执行 使用转移预测和数据流分析， 一些处理器让指令在程序实际执行之前就 “ 推 测执行” ， 并将结果存储在暂时的空间。 通过执行可能需要的指令， 可以使处理器的执行 机制尽可能地保待繁忙。

以上技术以及其他复杂的技术是实现处理器强大功能所必需的， 它们共同使得每个处理器周期执行许多指令成为可能，而不是每个指令需要很多周期。

性能平衡

当处理器的性能以惊人的速度向前发展的时候，计算机的其他关键部件并没有跟上。这引发了寻找性能平衡的需要：调整组成和结构，以补偿各种部件之间的能力不匹配。

由这种不匹配造成的问题在处理器和主存储器之间的接口上尤为突出。虽然处理器速度快速增长，但是主存储器和处理器之间的数据传输速度却差很多。处理器和主存储器之间的接口是整个计算机中最关键的路径，因为它负责在存储器芯片和处理器之间传送恒定的程序指令和数据流。如果内存或通道无法跟上处理器的需求，处理器将处于等待状态，并且会浪费宝贵的处理时间。

有许多系统结构的方法来解决这个问题，这些都体现在当代计算机的设计中。考虑下面的例子：

• 通过使DRAM的接口 ”更宽” 而不是 “更深", 以及增大总线的数据宽度， 来增加每次所能取出的位数。

• 通过在DRAM芯片中加人高速缓存。或其他缓冲机制来改变DRAM桵口 ． 使其更加 有效。

• 通过在主存和处理器之间引人更复杂、 更有效的高速缓存结构， 来减少存储器访问频度。这包括在处理器芯片中加人一级或者多级高速缓存， 以及在靠近处理器芯片的地方加入片外高速缓存。

• 通过使用高速总线和分层总线来缓冲和结构化数据流， 从而增加处理特与存储器之间相互连接的带宽。

另一个设计焦点从I/O设备的处理。 由于计算机变得更快、 更强大 ， 人们开发出更加复杂的 应用来支持使用要求频繁I/O橾作的外设。 图2.1给出个人计算机和工作站使用的一些典型例子 这些设备要求很高的数据吞吐量。 虽然目前的处理器能够处理这些设备输出的数据， 但在处理器和外设之间传送数据仍存在着问题。 这里的策略包括缓冲和暂存机制 ， 以及使用高速互连 和更为精巧的互连结构。 此外， 使用多处理器配置有助于满足I/O的需要。

所有问题的关键是平衡。设计者始终努力平衡处理器部件、主存储器、I/O设备及互连结构的吞吐量和处理要求。设计必须不断更新，以应付两个始终变化的因素。

• 从一种类型的元器件到另一种类型的元器件， 对于各种不同的技术领域（处理器、 总线， 存储器、 外设） ， 它们性能提高的速率相差很大。

• 新的应用和新的外围设备根据典型指令的描述和数据访问模式不断改变对系统特性的 要求。

因此，计算机设计是一种不断演变的艺术。本书试图呈现这种艺术形式所依赖的基础，一级这种艺术的当前发展状况。

芯片组成和体系结构的改进

当设计人员努力解决处理器性能与主存储器及其他部件的平衡问题时，提高处理器速度的 需求仍在增长。 有三种办法可实现处理器的提速：

． 提高处理器硬件速度。这个提速基本上要归功于处理器芯片上逻辑门的尺寸减小， 以便 更多的门能更紧密地组装在一起． 也要归功于时钟频率的提升。 随着门电路更紧密地集 成在一起， 信号的传播时间显著地降低， 从而允许处理器提速。 时钟获率的提升意味着每个操作被更迅速地执行。

． 提向插人在处理特和主存之间的each, 容扭和速度。 尤其是， 将处理器芯片自身的一部分用做cache，cache的存取时间会显著降低。

． 改变处理器的组成和体系结构以提高指令执行的有效速度。 典型地， 这包含使用各种形式的并行性。

传统上， 性能增益的主导因素是时钟速度的提升和逻辑密度的提高。然而， 随着时钟速度和逻辑密度的提高， 几个障碍变得更加显著[ INTE04b].

• 功耗：随着芯片上逻辑密度和时钟速度的提高， 芯片消耗的功率密度（瓦/cm')也随之提高。 高密 、 高速芯片的散热闲难成为一个重要的设计问题([GIBB04]. [BORK03]) 。

• RC延迟，电子在芯片上各晶体管间流动的速度受限于连接它们的金属线的电阻和电容。 特别是， 延迟随RC之积的增长而增长。 由于芯片上元件尺寸变小， 互连线更细 ， 从而电阻增加了，同时， 线排列更紧密， 电容也增大了。

• 存储器滞后和吞吐量：如前所述，存储器速度（延迟）和传输速度（吞吐量）滞后于处理器速度。

于是， 这里更强调以组成和体系结构的办法来改善性能。本书的后几章将讨论这些技术。

除了通过简单地提高时钟速度所实现的性能提升之外， 从20世纪80年代后期开始并一直持续了约15年， 为进一步提升性能还采用两种主要策略。第一， 增加cache容量。现在，处理器与主存之间一般都有两级或三级cache。由于芯片密度的提高. 更多的cache存储器已集成到处理器芯片上， 从而允许更快的cache访问。例如， 最初Pentium芯片将大约百分之10的芯片面积用于cache,, 而现在的芯片将大约一半多的芯片面积用于cache。通常，另外一半的四分之三用于管线相关的控制和缓冲。

第二， 处理器内指令执行逻耜变得越来越复杂， 以允许处理器内指令并行执行。两个值得重视的设计办法是流水化和超标量化。指令流水线的工作情况非常类似于制造厂的装配线， 它允许不同指令的不同执行段在流水线上同时工作。本质上讲， 超标量办法是允许在单个处理器内有多条指令流水线， 以便彼此无关的指令能并行地执行。

到90年代中后期，这两种策略正在到达收益递减点。当代处理器的内部组织已是非常复杂， 井能够从指令流中压榨出大量的并行性。看起来， 在这个方向上进一步显著地提升性能是相当有限的[GIBB04] 。随着处理器芯片设置三级cache，每级都有相当大的容量， 看来增加cache容量所带来的好处也达到了瓶颈。

然而， 简单地依靠提高时钟频率来提高性能又走入了已指出的功率悄耗问题。时钟频率越快，消耗的功率就越大，并且将达到某些基本的物理限制。

图2.2说明了我们一直在讨论的概念。最上面一行显示，按照摩尔定律单个芯片上的晶体管数量呈指数增长。同时，为防止功率进一步上升，时钟速度已经趋于平稳。 为了不断提高性能，设计人员除了开发更复杂的处理器之外，还必须寻找利用不断增长的晶体管数量的方法。近年来的成果是多核计算机芯片的发展。

考虑到上节提出的那些困难，设计者已转向一种根本性的新方法来改善性能：在同一芯片上安排多个处理器并带有大的共享cache。 同一芯片上多个处理器的使用， 也称为多核(multiple cores或multicore), 提供了无需提高时钟频率就能提高性能的潜力。 已有研究指出 ， 在处理器内 ， 性 能的提高大致正比于复杂度提高的平方根[ BORK03] 。 但是， 如果软件能够有效地支持多个处 理器的使用． 则处理器数目的加倍几乎使性能加倍。 因此， 此策略是使用芯片上两个较简单的处 理器 ， 而不是一个更复杂的处理器。

此外， 具有两个处理器 ， 一个更大的cache也是恰当的。 因芯片上存储逻辑的功耗远小于处 理逻辑的功耗， 故这种安排很重要。

随着芯片上的逻辑密度不断增大，单芯片上更多内核和更多缓存的趋势仍在继续。 双核芯片之后紧接着四核芯片，然后是8，然后是16，等等。 随着缓存量的增大，在芯片上创建两级和三级缓存的意义重大，最初，第一级缓存专用于单个处理器，第二级和第三级缓存专用于所有处理器。 二级缓存现在通常也是内核专用的。

芯片制造商现在正在每个芯片的内核数量上取得巨大飞跃，每个芯片的内核数量超过50个。在性能上的飞跃以及开发软件以利用如此大量的内核所面临的挑战产生一个新词汇：许多集成核（MIC）。

多核和MIC策略涉及单一芯片上的通用处理器的同质集合。与此同时，芯片制造商正在寻求另一种设计方案：一种具有多个通用处理器和图形处理单元（GPU）的芯片，以及用于视频处理和其他任务的专用内核。广义而言，GPU是一种用于在图形数据上执行并行操作的内核。它一般在插件图形卡(显示适配器)上，用于编码和渲染2D和3D图形以及处理视频。

由于GPU对多组数据执行并行操作，因此它们越来越多地被用作需要重复计算的各种应用程序的向量处理器。这模糊了GPU和CPU之间的界限[AROR12，FATA08，PROP11]。当这种处理器支持大量应用程序时，将使用术语“GPU上的通用计算”（GPGPU）。

我们将在第18章和第19章探讨多核计算机的设计特性以及GPGPU。

2.3两个定律：Ahmdahl定律和Little定律

本节我们看两个方程，称之为“定律”。这两个定律是无关的，但它们都能够加深我们对并行系统和多核系统性能的理解。

Ahmdahl定律

计算机系统设计人员寻求希望通过技术进步或设计变更来提高系统性能。比如使用并行处理器，使用内存缓存层次结构，以及由于技术改进而提升的内存访问时间和I / O传输速率。在这些例子中，我们要知道技术或设计的一方面的加速并不会导致相应的性能改进。 Amdahl定律简洁地表达了这种限制。

1967年Gene Amdahl首次提出Amdahl定律（[AMDA67]，[AMDA13]），并且使用多个处理器来处理程序的潜在加速。考虑运行在单个处理器上的程序，使得执行时间的一部分（1-f）包含固有顺序的代码，并且包含无限可并行且无调度开销的代码部分f。设T是使用单个处理器的程序的总执行时间。然后，使用带N个处理器的并行处理器加速完全利用程序的并行部分的速度如下所示：

使得执行时间的一部分（1～f）包含固有顺序的代码，以及包含无可调度开销的可无限并行化的代码的分数F。让T是使用单个处理器的程序的总执行时间。然后使用具有N个处理器的并行处理器，加速比如下：

加速比 = 单个处理器的执行时间/N个并行处理器的执行时间

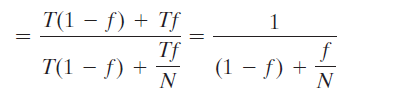


图2.3和2.4解释了这个等式，并得出如下两个重要结论：

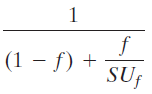
1. 当f很小的时候，并行处理器几乎不起作用。
2. 当N接近于无穷时，加速比无限接近1/（1-f），因此使用更多处理器的效率递减。

[GUST88]首次提出这些不太乐观的结论。 例如，服务器可以维护多个线程或多个任务来处理多个客户端，并且在处理器数量的限制内并行执行线程或任务。 许多数据库应用程序都涉及大量数据的计算，这些数据可以分解为多个并行任务。 然而，Amdahl定律说明了业界在开发具有多核机器时所面临的问题：在这类机器上运行的软件必须适应高度并行的执行环境，以利用并行处理的能力。

Amdahl定律可以用来评估计算机系统的任何设计或技术改进。 考虑会影响加速比的任何系统功能的增强。 加速可以表示为

加速比=提升前的性能/提升后的性能=提升前的执行时间/提升后的执行时间

假设在执行期间系统的一个功能在增强之前被使用了一小部分时间f，并且增强之后该功能的加速比是SUf。 那么系统的整体加速是

加速比=

例2.1假设某任务大量使用浮点操作，并且浮点操作占用了40％的时间。 采用新的硬件设计后，浮点模块加速了K倍。那么整体加速比如下：

加速比=

因此，与K无关的最大加速比是1.67。

Little定律

Little定律[LITT61, LITT11]是一个广泛应用的基本定律，它几乎可以用在任何统计上处于稳定状态的系统，并且没有漏洞。具体来说，我们有一个稳态系统，单位时间内物体到达的平均速率为λ，在系统中停留的平均时间为W。最后，在任何一个时间系统中物体的平均数量为L。Little定律将这三个变量的关系表示为L= λW。

用排队论的话来说，Little定律适用于排队系统。该系统的核心部分是一个为到达系统的物体提供服务的服务器，或称为窗口。如果窗口空闲，则到达的物体可以立刻被服务。否则就要排队。队列可能是单窗口单队列，多窗口单队列或多窗口多队列。当窗口服务完一个物体后，该物体离开。如果队列中还有物体正在等待，则立即将其中一个分派给窗口。此模型中的窗口可以代表任何为某物体集执行功能或提供服务的东西。比如处理器为进程提供服务;传输线为数据包或数据帧提供传输服务;I / O设备为I / O请求提供读写服务。

下面的论述有助于我们理解Little公式，以单个物体为例，当它到达时发现在它前面平均有L个物体，一个正在服务，其余的在队列中。因为L定义为等待的的平均数量，所以当该项目服务完成离开系统时，系统中平均保留相同数量的物体，即L个。此外，物体在系统中的停留的平均时间是W，因为物体以λ的速率到达，我们可以推断在时间W中，总有λW个物体到达。因此w = λW。

总之，在稳态条件下排队系统中物体的平均数等于物体到达的平均速率乘以物体在系统中停留的平均时间。 这个关系几乎不需要假设。 我们不需要知道服务时间的分布，到达时间的分布，以及被服务物体的顺序或优先级。 由于其简单性和通用性，Little定律非常有用，并且由于对多核计算机相关性能问题的兴趣而经历了某种程度的复兴。

[LITT11]的一个简单例子说明了如何应用Little'定律。考虑一个多核系统，每个内核可执行多个线程。在某种程度上内核共享内存。内核不仅共享主存，通常也共享缓存。当一个线程正在执行时，可能会出现需要从公共内存中检索数据的情况。线程停止执行并发出对该数据的请求。所有这些停止的线程都在队列中。如果系统被用作服务器，分析人员可以根据用户请求的速率确定系统需求，然后将其转化为对已有线程进行数据请求的速率，以响应单个用户请求。为此，每个用户请求都被分解为以线程形式实现的子任务。然后，我们得出λ=在所有成员的请求被分解成详细子任务之后所需的总的线程处理的平均速率。将L定义为在相关时间内等待的线程的平均数。W 为平均响应时间。这个简单的模型可以作为设计者判断用户需求是否得到满足的指导，如果没有，则提供必要的改进量的度量。

2.4计算机性能的基本评价

在性能处理器硬件和设置新系统的需要时，性能是必需考虑的关键因素之一，这包括成本、尺寸、安全性、可靠性以及某些情况下的能源消耗。

在不同的处理器之间、 甚至在同一系列的处理器之间进行意味深长的性能比较是困难的。当执行一个给定的应用时， 未加工的速度指标远不及处理器如何完成任务来得重要。 不幸的是， 应用的执行并不仅仅取决于处理器的速度， 还依赖于其指令集 、 实现语言的选择、 编译器的效率以及实现该应用的编程技巧。

本节首先介绍一些处理器速度的传统测量方法， 下节我们研究基准测试，这是评估处理器和计算机系统性能的最常用方法。然后讨论如何从多个测试中获得平均结果。

时钟速度

处理器执行的操作， 例如取指令、 译码该指令 、 执行算术运算等， 都是由系统时钟掌控的。 典型的做法是 ， 所有操作都随着时钟的脉冲开始。 因此 ， 在最基本的级别 ， 处理器的速度由时钟 产生的脉冲频率来指示， 用每秒周期数或赫兹(Hz)来测量。

一般情况下，时钟信号由水晶振子产生，水晶振子在有动力供应时能产生一个连续的信号波。该波被转化为一个数字电压脉冲流，连续地供应给处理器电路（如图2.5所示）。例如，一个1GHz的处理器每秒接受10亿个脉冲。 脉冲的速率被定义为时钟频率， 或时钟速度。 每增加一个脉冲或时钟被称为一个时钟周期， 或时钟滴答声。 两个脉冲之间的时间定义为周期时间。

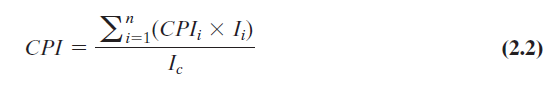
时钟频率不是任意的， 它必须适应处理器的物理层。 处理器中的操作需要信号将其从处理

器的一个元件传达到另一个元件。 当信号被放在处理器内部的一根线上时， 它将占用一些有限的时问量来使电压水平平静下来 ， 以便一个正确的值(1或0)可用。 此外 这取决于处理器电路的物理层 ， 有些信号可能比其他信号变化得更快， 因此， 操作必须同步， 以便适当的电信号（电压）值可为每个操作用到。

指令的执行包含很多离散的步骤。 例如 ， 从存储器中取出该指令 、 译码指令的各个部分、取和存数据以及执行算数和逻辑运算，因此， 大多数处理器的大部分指令需要多个时钟周期来完成。 打些指令可能只需要几个周期， 而另一些指令需要几十个周期。 此外， 当使用流水线时 ， 多条指令被同时执行， 因此． 不同处理器的时钟速度的直接比较是不能说明性能的整体情况的。

指令执行速度

处理器由时钟驱动｀ 时钟具打固定的频率f, 或等价为固定的时钟周期τ, 这里τ= 1/f。 定义一个程序的指令条数l，为运行完该程序所执行的机器指令条数。 注意这是指令执行的条数， 而不是该程序目标代码中的指令条数。 程序的一个重要参数是每条指令的平均周期数(average cycles per instruction, CPI) 。 如果所有指令需要相同的时钟周期数， 则该程序的CPI就是一个固定值。 然而， 对于任意指定的处理器， 不同的指令类型 ， 例如取数、 存数、 分支等等 ， 会需要不同的时钟周期数。 如果用CPIi来表示指令类型i所需要的周期数， 用Ii表示在某一给定程序中所执行的i类指令的条数， 则我们可以计算整个CPI如下:



处理器执行一个给定的程序所需要的时间可以表示为：

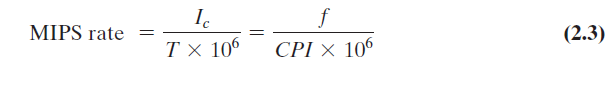


我们可以通过分析一条指令的执行过程来定义该公式， 指令执行的一部分工作是由处理器完成的， 而另一部分时间花费在处理器与存储器之间的字传送上。 在后一种情况中， 传送时间取决于存储器周期时间， 它可能比处理器周期时间长。 将上等式改写成：



这里 ， p是译码和执行指令所需要的处理器周期数， m是所需的存储器访问次数， k是存储器周期时间和处理器周期时间之比。 上面等式中的5个性能因子(/c、 p、 m、 k、 τ)受4个系统属性影响：指令集设计（亦称指令集结构）、编译技术（编译器如何高效地将高级语言程序转换为有效的机器语言程序）、处理器实现以及cache与主存的层次结构。表2.1是一个矩阵， 其横轴表示5个性能因子， 纵轴表示4个系统属性。 单元中的X说明系统属性影响着性能因子。

处理器性能的一个通用度量是指令执行的速率 ． 表示成每秒百万条指令(MIPS). 也称为 MIPS速度。 我们可以用时钟频率和 CPI 表示MIPS速率如下：



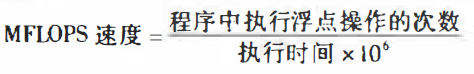
例2.2 考虑在一个400MHz的处理器上运行一个包含200万条指令的程序，该程序由四种主

要的指令类型组成。 基于程序踪迹实验的结果， 得出的指令混合和 每一种指令类型的 CPI如下：

|  |  |  |
| --- | --- | --- |
| 指令类型 | CPI | 指令混合比（%） |
| 算数和逻辑 | 1 | 60 |
| cache命中的取数/存数 | 2 | 18 |
| 分支 | 4 | 12 |
| cache失效的存储器访问 | 8 | 10 |

当由单一处理器执行该程序时， 其平均CPl=0.6+(2x0.18) +(4x0.12 ) +(8x0.1) =2.24, 相应的MIPS速度:(400 X 10^6)/(224 x 10^6)≈178。

另一个通用的性能度量仅仅用于浮点指令，这在很多科学计算和游戏应用中常见。浮点性能表示为每秒百万条浮点操作（MFLOPS），定义如下：



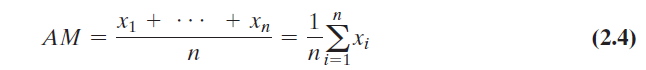
2.5计算平均值

在评估计算机系统性能的某些方面时，通常会使用单一的数字（例如执行时间或内存消耗）来表征性能并比较系统。显然，单个数字只能提供一个非常简化的系统功能视图。尽管如此，在比较性能时特别是在基准测试领域仍然通常使用单个数字[SMIT88]。

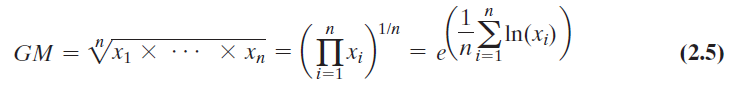
如2.6节讨论的那样，我们使用基准来比较系统需要计算与执行时间相关的一组数据点的平均值。事实证明，有多种算法可用于计算平均值，这一直是基准测试领域中一些争议的根源。在本节中，我们将定义这些替代算法并对它们的一些属性进行解释。为我们在基准测试中的下一节平均值计算做了准备。

用于计算均值的三个常用公式是算术，几何和谐波。给定n个实数（x1，x2，...，xn），这三种方法的定义如下：

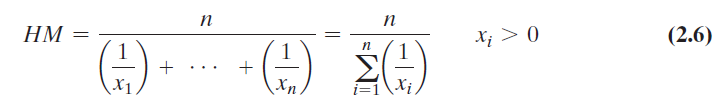
算数平均值：



几何平均值：



谐波平均值：

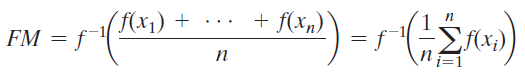


下列不等式成立：



当且仅当x1=x2=...=xn时等号成立。

我们可以通过定义函数平均值来深入了解这些计算。令f(x)是一个在0 <y <∞区间连续的单调函数。对于n个正实数(x1, x2，…，xn)的函数f(x)的函数均值定义为

函数均值（Functional mean）

其中是f(x)倒数，方程（2.1）到（2.3）中定义的平均值是函数均值的特例，如下所示：

算数均值（AM）是关于f(x)=x的函数均值

几何均值（GM）是关于f(x)=lnx的函数均值

谐波平均值（HM）是关于f(x)=1/x的函数均值

例2.3 图2.6说明了应用于各种数据集的三种方法，每种数据集有11个数据点和11个最大数据点值。中值也包含在图表中。也许在这个图中最引人注目的是，当数据向大值倾斜或有小值异常值时，HM有产生误导结果的可能。

我们现在考虑一下，这些方法中哪一种适合给定的性能度量。作为这些评定的序言，应该指出的是，许多论文([CITR06]、[FLEM86]、[GILA95]、[JACO95]、[JOHN04]、[SMIT88])和书籍([mashn12]、[HWAN93]、[JAIN91]、[LILJ00]经过多年的分析得出了矛盾的结论。为了简化一个复杂的争论，我们只是注意到，得出的结论很大程度上取决于所选择的例子和表述方式。

算术平均值

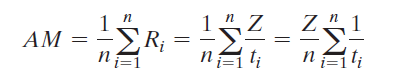
如果所有的测量值都是有意义的，那么算术平均值（AM）就是一个合适的度量。 AM是比较系统执行时间的好选择。例如，我们想使用系统来进行大规模的仿真研究并希望评估出几种可选的替代产品。在每个系统上，我们可以多次使用不同的输入值来运行仿真，然后选取平均时间。使用多个不同输入的运行结果时应确保结果不会因给定输入集的一些特殊特征而产生严重偏差。每次运行的AM都是模拟系统性能的良好指标，并且可用于系统比较。

算术平均值用于一个基于时间的变量(例如秒)，比如程序执行时间，它与总时间成正比。如果总时间翻倍，均值翻倍。

谐波平均值

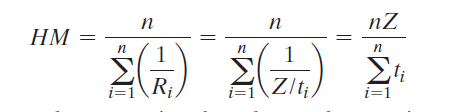
在某些情况下，系统的执行率被看作对系统性能更有用的度量。比如指令执行速率(用MIPS或MFLOPS表示)，或者是程序执行速率(以度量给定类型的程序执行速率)。考虑一下我们希望如何表示计算的平均数。首先，希望平均速率与总速率成比例是没有意义的，总速率被定义为单个速率的和。利率的总和是一个毫无意义的统计数字。相反地，我们希望平均值与总执行时间成反比。例如，如果在一组程序中系统C执行所有基准程序的总时间是系统D的两倍，那么我们希望系统C的执行速度的平均值是系统D的一半。

先来看一个基础例子，我们首先研究一下如何计算AM。假设我们有一组n个基准程序，并将给定系统上每个程序的执行时间记录为t1，t2，...，tn。 为了简单起见，让我们假设每个程序执行相同数量的操作Z; 我们可以对各个程序进行加权并进行相应的计算，但这不会改变我们论点的结论。 每个程序的执行率为Ri = Z / ti。 我们使用AM来计算平均执行率。



我们看到AM的执行速度与执行时间的倒数成正比，这和执行时间的和成反比是不一样的。因此，AM没有所需的性质。

HM的结果如下。



HM与总执行时间成反比，这是我们所需要的。

例2.4 我们用一个简单的数值例子说明计算均值的两种方法之间的差异，如表2.2所示。该表比较了三台计算机执行两个程序的性能。为了简单起见，我们假设每个程序执行108个浮点运算。表格的左半部分显示了每台计算机运行每个程序的执行时间，总执行时间以及执行时间的AM。计算机A执行的总时间少于B，B少于C，AM准确反映了这一关系。

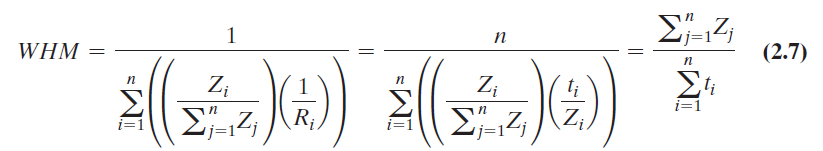
表中右半部分比较了执行速率，用MFLOPS表示。计算速率很简单。例如，程序1执行1亿浮点运算。计算机A需要2秒，则速率为100/2 = 50。接下来，考虑速率的AM。计算机A的值最大，这表明A是最快的计算机。就总的执行时间而言，A的用时最少，所以它是三者中最快的。但是速率的AM显示B比C慢，而实际上B比C快。再看看HM的值，我们发现它们正确地反映了计算机的速度顺序。这证实了在计算速率时，HM是首选。

读者可能想知道我们为什么要做这些努力。如果我们想比较执行时间，我们可以简单地比较三个系统的总执行时间。如果我们想比较速率，我们可以取总执行时间的倒数，如表所示，进行单独计算有两个原因，而不是仅查看总数字：（缺表）

1.客户或研究人员可能不仅对总体的平均性能感兴趣，还对不同类型的基准程序(如业务应用程序、科学建模、多媒体应用程序和系统程序)的性能也感兴趣。因此，需要根据基准的类型进行分类，并提供总数。

2.通常用于评估的不同程序的权重是不同的。在表2.2中，假设两个测试程序执行相同数量的操作。如果不是这样的话，我们可能需要相应的加权。或者不同的程序可以用不同的权重来反映重要性或优先级。

下面我们来看如果测试程序与操作的数量成比例，结果会是什么。按照前面的符号，任一程序i在时间ti中执行Zi条指令。每个速率都由指令计数加权。因此，加权HM为:



可以看到加权HM是运算计数和除以执行时间之和的商。

几何平均数

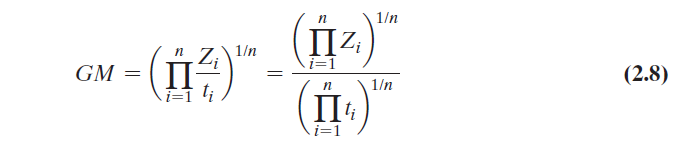
从这三种方式的方程式来看，相对于GM我们可以更直观地了解AM和HM的过程。首先，我们注意到，对于值的变化，GM对数据集中的所有值都给予相同的权重。

例如，一个要求其平均数的数据集合包括几个大值和小值。这里的AM受大值影响较大。最大值10%的变化就会对其产生明显的影响，而最小值产生这些变化带来的影响则可以被忽略。相比之下，任何数据值10%的变化都会导致GM的变化:

例2.5 图2.6中的数据集（e）对这点做了说明。 以下是将数据集中的最大值和最小值增加10％的结果：

|  |  |  |
| --- | --- | --- |
|  | GM | AM |
| 原始值 | 3.37 | 4.45 |
| 最大值从11增到12.1（+10%） | 3.40（+0.87%） | 4.55（+2.24%） |
| 最小值从1增到1.1（+10%） | 3.40（+0.87%） | 4.46（+0.2%） |

第二个观察结果是，对于速率的GM来说，速率的GM等于GMs的速率:



将此等式和2.4式比较。

对于执行时间，与速率相反，相对于更加直观的AM来说，GM的一个缺点它可能是非单调的。换言之，可能存在一个数据集，它的AM大于另一个集合的AM，但是GM较小。

例2.6 在图2.6中，数据集d的AM大于数据集c的AM，但是GM则相反。

|  |  |  |
| --- | --- | --- |
|  | 数据集c | 数据集d |
| AM | 7.00 | 7.55 |
| GM | 6.68 | 6.42 |

GM对于基准分析的吸引力之一是它在衡量机器的相对性能时提供一致的结果。 实际上，基准测试主要用于比较机器的性能指标。 正如我们所看到的，结果是标准化为参考机器的值来表示的。

例2.7 一个简单的例子将说明GM对标准化结果展示一致性的方式。 在表2.3中，我们使用了与表2.2中相同的性能结果。 在表2.3a中，所有的结果都被归一化到计算机A，并且这些平均值根据归一化的值进行计算。 基于总执行时间，A比B快，这比C快。归一化时间的AM和GM都反映了这一点。 在表2.3b中，系统现在归一化为B.GM再次正确地反映了三台计算机的相对速度，但现在AM产生了不同的排序。一致性并不总是产生正确的结果。 在表2.4中，一些执行时间被改变。 AM再次报告两种标准化的结果不一致。 全球机制报告了一致的结果，但结果是B比A和C更快，它们是相等的。

正是这样的例子引发了前面提到的“基准意味着战争”。可以肯定地说，没有单个数字能够提供比较系统性能所需的所有信息。然而，尽管在文献中有一些相互矛盾的观点，SPEC还是选择使用GM，原因如下：（缺四个表）

1.如前所述，无论使用哪个系统作为参考，GM的结果都是一致的。因为基准测试主要是做比较分析，这是一个重要的特性。

2.如[MCMA93]中所记录的，相对于HM和AM，GM不容易受离群值的影响，这一点在SPEC分析师[MASH04]的后续分析中得到证实。

3.所示。[MASH04]证明，由于归一化数的普遍偏态分布，对数正态分布比正态分布能更好地模拟性能比。这在[CITR06]中得到证实。而且，如方程（2.5）所示，GM可以被描述为对数正态分布的回归平均值。

2.6基准和规格

基准原则

使用像MIPS和MFLOPS一祥的度从来评价处理器的性能已经被证明是不充分的。 因为不同的指令集． 所以指令执行速率不是很好地比较不同体系结构性能的方法。 例如， 打虑如下高级语言语句：

A=B+C /\*假设所有变量在主存中\*/

传统的指令集结构是指复杂指令集计算机(CISC), 该语句能够被编译成一条处理器指令：

add mem(B),mem(C),mem(A)

而在典型的RISC机上，该语句可能被编译成：

load mem(B),reg(1);

load mem(C),reg(2);

add reg(1),reg(2),reg(3);

store reg(3),mem(A)

因为RISC 结构的特性（将在第15章中讨论） ， 所以两种机器可能花费控不多相同的时间运 行最初的环级由言语句。如果这个例子是两种机器的代表， 则若CISC 机的速度为IMIPS, 那么 HISC机的速度将丛4MIPS。 但二者处理同拭的高级语言程序所需的时间相同。

另一方面的考虑是， 处理器执行某一给定程序的性能并不能决定它将如何执行其他类型的应用程序。于是， 从20世纪80年代后期和 90年代早期开始， 业界和学术界壮欢使用一系列基准程序来测R系统的性能。 一组相同的程序可以运行在不同的机器上， 并对执行时间进行比较。基准为想要购买系统的客户提供指导，并且可以帮助供应商和设计人员确定如何设计符合基准目标的系统。

文献[WEI C90]列出了作为基准程序所需要具备的一些特征

())它由环级由言编写， 可以方便地应用于不同的机器。

1. 它是各种特殊程序设计方式的代表， 例如， 系统程序设计 ， 数字程序设计或商业程序设计

(3)易于度仗。

(4 )它打广泛的发行。

SPEC基准程序

业界、学术界和研究院对公认的计算机性能衡量方法的共同需求导致了标准基准程序集的

发展。 基准程序织就怂一个程序如合 ． 使用环级语言定义 ， 它试图对在特殊应用或系统程序设计领域中的计订．机提供一种有代表性的测试。 Jil茗名的测试程序集由系统性能评估公司(SPEC,一种T一业社团）定义和维护。这个组织为评估计算机系统定义了几个基准套件。 SPEC性能测试广泛应用千比较和研究的目的。

最著名的SPEC基准程序如是SPECCPU2006, 这是一种测肚强调处理器应用的工业标准其也就姑说. SPEC CPU2006适合于测试计饵密集型应用而非VO密集型应用的性能。

其他SPEC测试集如下：

SPECviewperf：基于专业应用程序测量3D图形性能的标准。

SPECwpc：衡量基于多种专业应用（包括媒体和娱乐，产品开发，生命科学，金融服务和能源）的工作站性能的所有关键的基准。

SPECjvm2008：旨在评估Java虚拟机（JVM）客户端平台的组合硬件和软件方面的性能。

SPECjbb2013（Java业务基准）：评估服务器端基于Java的电子商务应用程序的基准。

SPECsfs2008：旨在评估文件服务器的速度和请求处理能力。

SPECvirt\_sc2013：虚拟化服务器合并中使用的数据中心服务器的性能评估。测量所有系统组件的端到端性能，包括硬件，虚拟化平台以及虚拟客户操作系统和应用程序软件。该基准测试支持硬件虚拟化，操作系统虚拟化和硬件分区方案。

CPU2006套件是基于已有的应用程序，它们已经被SPEC行业成员移植到了各种各样的平台上。 为了使基准测试结果可靠和真实，CPU2006的基准测试结果来自实际应用，而不是使用人工循环程序或合成基准测试。 该套件包含用C和C ++编写的12个整型基准，以及用C，C ++和Fortran编写的17个浮点基准（表2.5和2.6）。该套件包含超过300万行代码。 这是来自SPEC的第五代处理器密集型套件，替代了SPEC CPU2000，SPEC CPU95，SPEC CPU92和SPEC CPU89 [HENN07]。

为了更好地理解使用CPU2006发布的系统结果，我们在SPEC文档中定义了以下术语:

基准测试：用高级语言编写的程序，可以在任何实现编译器的计算机上编译和执行。

被测系统：这是要评估的系统。

推理机：SPEC使用该系统为所有基准建立基线性能。在此机器上运行和度量每个基准，以建立该基准的参考时间。通过运行CPU2006基准测试并比较在推理机上运行相同程序的结果，对正在测试的系统进行评估。

基准指标：这些指标对所有报告的结果都是必须的，并且有严格的编译指南。实质上，应该在每个测试系统上使用带有或多或少默认设置的标准编译器来实现类似的结果。

峰值度量：这使用户可以尝试通过优化编译器输出来优化系统性能。例如，可以在每个基准测试中使用不同的编译器选项，并允许反馈指导优化。

速度指标：仅衡量执行编译基准所花费的时间。速度度量用于比较计算机完成单个任务的能力。

速率度量：衡量计算机在一定时间内能完成多少任务的度量，被称为吞吐量，容量或速率度量。速率度量允许被测试的系统利用多个处理器执行同步任务。

SPEC使用历史悠久的Sun系统“Ultra Enterprise 2”，该系统于1997年引入，作为推理机器。 该推理机器使用296-MHz UltraSPARC II处理器。 CPU2006参考机器上的CINT2006和CFP2006的基准指标运行需要大约12天的时间。表2.5和2.6显示了使用参考机器运行每个基准的时间量以及参考机器上的动态指令计数，如[PHAN07]中所报告的。 这些值是每个程序运行期间执行指令的实际数量。

我们现在考虑评估系统的具体计算。 我们考虑整数基准; 并使用相同的过程来创建浮点基准值。 对于整数基准测试，测试套件中有12个程序。 计算是一个三步过程（图2.7）：

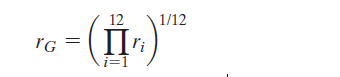
1. 评估测试系统的第一步是编译运行三次系统上的每个程序。对于每个程序，测量运行时间并选择中值。使用三次运行并取中值的原因是要考虑到不是程序固有的执行时间的变化，例如磁盘访问时间的变化，以及OS内核执行的变化。（缺图2.7）

2.接下来，通过计算参考运行时间与系统运行时间的比值，比值的计算如下:



其中，Trefi是基准程序i在参照系统上的运行时间，而Tsuti是基准程序i在被测系统上的运行时间。因此对于速度更快的机器来说，比值更高。

3.最后计算12个运行时比值的几何平均值，得出总体指标:



对于整数基准，可以计算四个指标:

SPECint2006:在使用峰值调优编译基准时，12个标准比值的几何平均值。

SPECint\_base2006:使用基本调优编译基准时12个标准比值的几何平均值。

SPECint\_rate2006:在使用峰值调优编译基准时，12个标准化吞吐量比的几何平均值。

SPECint\_rate\_base2006:使用基本调优编译基准时，12个标准化吞吐量比的几何平均值。

例2.9 表2.7a中显示了Sun Blade 1000的结果。 SPEC CPU2006整数基准之一是464.h264ref。 这是最新的最先进的视频压缩标准H.264 / AVC（高级视频编码）的可参考的实现。 Sun Blade 1000以5,259秒的中间时间执行此程序。 参考机器实现需要22,130秒。 该比值计算为：22,130 / 5,259 = 4.21。速度度量通过取比率乘积的根号12来计算：

（3.18 x 2.96 x 2.98 x 3.91 x 3.17 x 3.61 x 3.51 x 2.01 x 4.21 x 2.43 x 2.75 x 3.42）(1/12次方) = 3.12

速率度量考虑了具有多个处理器的系统。要测试一台机器，需要复制N个——通常这等于测试系统上的处理器数量或同时执行的线程数量。每个单独的测试程序的速率是由三次测试的中位数决定的。 每次运行由在测试系统上同时运行的程序的N个副本组成。执行时间是完成所有副本所需的时间(即，从第一次复制开始到最后一次复制完成的时间)。该程序的速率度量由以下公式计算:



被测系统的速率得分由测试套件中每个程序的速率的几何平均值决定。

例2.10 表2.7b中显示了Sun Blade X6250的结果。 该系统有两个处理器芯片，每个芯片有两个内核，共有四个内核。 为了获得速率指标，每个基准程序在四个内核上同时执行，执行时间是从四个副本开始到最后运行结束的时间。 速度比率如前所述计算，速度值仅为速度比率的四倍。 最终的速率指标是通过计算速率的几何平均值得出的：

（78.63 x 62.97 x 60.87 x 77.29 x 65.87 x 83.68 x 76.70 x 134.98 x 106.65 x 40.39 x 48.41 x 65.40）1/12(1/12次方)=71.59（缺很多表）

2.7 关键词、思考题和习题

关键词

Amdahl’s law 阿姆达尔定律

arithmetic mean (AM) 算数平均值

base metric 基本度量

Benchmark 基准

clock cycle 时钟周期

clock cycle time 时钟周期时间

clock rate 时钟频率

clock speed 时钟速度

clock tick 时钟周期（时钟滴答）

cycles per instruction (*CPI*) 每条指令的周期数（CPI）

functional mean (FM) 函数均值

general-purpose computing 通用计算

on GPU (GPGPU) 通用图形处理器

geometric mean (GM) 几何平均数

graphics processing unit(GPU) 图形处理单元

harmonic mean (HM) 调和/谐波平均值

instruction execution rate 指令执行率

Little’s law Little定律

many integrated core (MIC)集成内核

Microprocessor 微处理器

MIPS rate 指令执行率

Multicore 多核

peak metric 峰值度量

rate metric 速率度量

reference machine参考机/推理机

speed metric 速度度量

SPEC规格

system under test 测试系统

throughput吞吐量

思考题

2.1列出并简要讨论当时钟速度和逻辑密度变大时出现的障碍。

2.2使用缓存的优点是什么？

2.3简要描述一些可以提高处理器速度的方法。

2.4简要描述阿姆达尔定律。

2.5定义时钟频速率，它是否与时钟速度相似？

2.6定义MIPS和FLOPS。

2.7调和平均值何时是系统性能的适当度量？

2.8解释每个与Little's定律有关的变量。

2.9什么是SPEC基准？

2.10使用SPEC基准评估系统的性能时，为什么每个程序都要编译运行三次？

习题

2.1一个基准测试程序首先在200 MHz上运行，然后在400 MHz处理器上运行。执行的程序包含200,000条指令，其中包含以下指令和时钟周期计数：

|  |  |  |
| --- | --- | --- |
| 指令类型 | 指令数量 | CPI |
| 整数运算 | 100000 | 1 |
| 数据传输 | 60000 | 2 |
| 浮点 | 30000 | 2 |
| 控制转移 | 10000 | 2 |

计算两种情况下有效的CPI和MIPS速率。

2.2有两个机器，两个不同的指令集，它们的时钟频率都是200mhz。在运行一组给定的基准程序的两台机器上，记录以下测量数据:

|  |  |  |
| --- | --- | --- |
| 指令类型 | 指令数量（百万） | CPI |
| A机器  运算逻辑指令  加载/存储指令  分支指令  其他指令 | 8  4  2  4 | 1  3  4  3 |
| B机器  运算逻辑指令  加载/存储指令  分支指令  其他指令 | 10  8  2  4 | 1  2  4  3 |

（1）计算每台机器的有效CPI，MIPS速率和执行时间。

（2）分析所得结果。

2.3 CISC和RISC设计的早期例子分别是VAX 11/780和IBM RS / 6000。 使用典型的基准测试程序，可以得出以下机器特性：

|  |  |  |  |
| --- | --- | --- | --- |
| 处理器 | 时钟频率（MHz） | 性能（MIPS） | CPU时间 |
| VAX11/780 | 5 | 1 | 12x |
| IBM RS/6000 | 25 | 18 | x |

最后一列表明VAX所需的时间比IBM时间长12倍。

（1）在这两台机器上运行的这个基准测试程序机器码的指令数量的相对大小是多少？

（2）这两台机器的CPI是多少？

2.4三台计算机上执行四个基准程序的结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | A计算机 | B计算机 | C计算机 |
| 程序1 | 1 | 10 | 20 |
| 程序2 | 1000 | 100 | 20 |
| 程序3 | 500 | 1000 | 50 |
| 程序4 | 100 | 800 | 100 |

该表显示的执行时间以秒为单位，每个程序执行100,000,000条指令。 计算每台计算机执行这些程序的MIPS值。 然后计算四个程序等权重的算术和谐波平均值，并根据算术平均值和谐波平均值对计算机进行排序。

2.5基于文献[HEAT84]中的数据，下表是三台机器上五个不同基准程序的执行时间（以秒为单位）。

|  |  |  |  |
| --- | --- | --- | --- |
| 基准程序 | 处理器 | | |
| R | M | Z |
| E | 417 | 244 | 134 |
| F | 83 | 70 | 70 |
| H | 66 | 153 | 135 |
| I | 39449 | 35527 | 66000 |
| K | 772 | 368 | 369 |

1. 计算每个基准处理器的速度指标，将其归一化到机器R，即R的比值为1.0。其他比率用公式(2.5)计算，R作为参考系统。然后用式(2.3)计算每个系统的算术平均值。这是[HEAT84]所采取的方法。

（2）用M作为参考机重复（1）部分。 在[HEAT84]中没有试过这个计算。

（3）根据前面两个计算中的任一计算，可以看出哪台机器最慢？

（4）使用公式（2.6）中定义的几何平均值，重复（1）和（2）的计算。 基于这两个计算，哪台机器最慢？

2.6为了阐明上述问题的结果，我们来看一个简单的例子

|  |  |  |  |
| --- | --- | --- | --- |
| 基准程序 | 处理器 | | |
| X | Y | Z |
| 1 | 20 | 10 | 40 |
| 2 | 40 | 80 | 20 |

（1）分别使用X、Y作为参考机，计算每个系统的算术平均值。 通常我们认为三台机器具有大致相同的性能，而算术平均值会产生令人误解的结果。

（2）分别使用X、Y作为参考机，计算每个系统的几何平均值。 该结果比算术平均值更真实。

2.7 考虑2.5节中关于计算平均CPI和MIPS速率的例子，计算得出CPI = 2.24，MIPS= 178。现在假设程序可以在8个并行任务或线程中执行，并且每个任务的指令数量大致相等。在一个8核系统上执行，每个内核（处理器）具有与原来使用的单处理器相同的性能。部件之间的协调和同步为每个任务增加了25,000个指令。 假设每个任务的指令组合与示例中的相同，但是由于内存争用，将缓存缺失的CPI增加到12个周期。

（1）确定平均CPI。

（2）确定相应的MIPS速率。

（3）计算加速因子。

（4）将实际加速因子与由Amdhal定律确定的理论加速因子进行比较。

2.8 处理器访问主内存的平均访问时间为T2。在处理器和主存储器之间插入一个较小的缓冲存储器。缓存的访问时间明显快于T1 6t2。缓存随时保存一些主存的拷贝，并且是在不久的将来最可能被访问的。假设处理器访问的下一个单元在缓存中的概率是H，则H为命中率。

（1）对于任何单个内存访问，理论上访问缓存而不是主存的加速比是多少?

（2）T为平均存取时间。将T表示为T1、T2和H的函数，整体的加速比（表示为H的函数）是多少?

（3）在实践中,一个系统可能是设计的处理器必须首先访问缓存来确定这个词是在缓存中,如果不是,那么访问主存,以便在一个小姐(相反),内存访问时间T1 + T2。将T表示为T1、T2和h的函数，现在计算加速比，并与(b)部分生成的结果进行比较。

实际上，可以设计一个系统使得处理器必须首先访问高速缓存以确定能否命中，如果没有则访问主存储器，那么在未命中时，存储器访问时间为T1 + T2。T为T1，T2和H的函数。现在计算加速并与（2）中产生的结果进行比较。

2.9 医院的一名医生观察到，平均每小时有6名病人到达，医院里通常有3名病人。 那么每个病人在医院的平均时间是多少？

2.10 通过图2.8a，我们可以更深入地了解Little's定律。 在一段时间T内，共有C项到达系统，等待服务并完成服务。 上面的实线表示到达的时间顺序，下面的实线表示出发的时间顺序。 由两条线包围的阴影区域表示系统以秒为单位完成的总“工作”; 令A为全部工作，我们希望推导出L，W和λ之间的关系。

（1）图2.8b将总面积分成水平矩形，每个矩形都有一个作业的高度。将所有这些矩形向左滑动，使它们的左线在t = 0处排成一行。写出一个与A，C和W相关的方程。

（2）图2.8c将总面积划分为垂直矩形，由虚线指示的垂直过渡边界定义。将这些矩形向下滑动，以使它们的下边缘在N（t）= 0处排列。写出一个与A，T和L相关的方程。

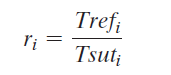
（3）从（1）和（2）的结果中推导出L =λW。（缺图）

2.11 在图2.8a中，工作时间t = 0,1,1.5,3.25,5.25和7.75。 相应的完成时间为t = 2,3,3.5,4.25,5.25和8.75。

（1）求出图2.8b中六个矩形中每一个的面积，并求和得到总面积A，写出详细步骤。

（2）求出图2.8c中10个矩形的每一个的面积，并求和以得到总面积A，写出详细步骤。

2.12 在2.6节中，我们定义了用于比较被测系统和参考系统的基准比率是：



（1）与参考系统相比，上述等式提供了被测系统加速比的计算。 假设在测试程序中执行的浮点运算的数量是Ii。 现在将加速比显示为指令执行速率FLOPSi的函数。

（2）使性能标准化的另一种方式是将系统的性能表示为相对于另一个系统的性能的百分比变化。 先将这种相对变化表示为指令执行速率的函数，然后再作为执行时间的函数。

2.13 假定基准程序在参考机器M1上以x秒执行。 相同的程序在系统M2，M3和M4上分别以0.75x，1.5x和0.4x秒执行。

（1）写出被测试的三个系统相对于M2的每个系统的加速比。

（2）写出三个系统的相对加速。 说明机器比较的三种方式（执行时间，加速比，相对加速比）。

2.14用机器M3作为参考机器重复上述计算。 这将如何影响四个系统的相对排名？

2.15用表2.4的计算机时间数据重新计算表2.2中的结果并说明。

2.16公式2.5说明了几何平均值的两种不同计算公式，一种使用乘积运算符，另一种使用求和运算符。

（1）证明这两个公式是等价的。

（2）为何计算几何平均值时使用求和公式？

2.17项目。2.5节列出了许多参考文件记录了“基准意味着战争”。所有参考论文可在box.com/COA10e获得。 阅读这些文件并总结支持和反对SPEC计算几何平均值的理由。

第三章

从顶层来看，计算机包括CPU（中央处理器）、存储器和I/O部件，每种类型有一个或多个模块。这些部件以某种方式互相连接 ， 实现计算机的基本功能， 即执行程1礼因此， 在硕层， 我们可以通过两种方法来描述计算机系统'(I)描述每个部件的外部操作， 即它与其他部件之间交换的数据和控制信号。(2)描述互连结构和管理互连结构所要求的控制。

从顶层考察结构和功能的项要性在于它有助干理斛计算机的特性， 另一个项要性在于它可 以用来理解性能评估这一日趋复杂的问题。 只要把捏顶层的结构和功能， 就可以洞察系统的瓶颈 选择其他杆代通路， 了解因部件失效而导致的系统故店的程度， 并容易提升系统的性能。 在许多情况下 ， 只有通过修改设计而不仅仅是提高单个部件的速度和可靠性， 才能满足对更大的 系统功能和故体安全能力的要求。

本章重点讨论用于计算机部件互连的基本结构。 作为背景， 本章首先简要考察基本部件及 其接口要求， 然后提供功能概述 ， 最后论述如何使用总线来连接系统部件。

3.1 计算机的部件

正如第1章所述，几乎所存的当代计算机设计都是以普林斯桥几级研究院的冯. 诺伊曼提出的概念为基础。 这种设计称为冯·诺伊曼结构， 它基于以下3个主要概念：

• 数据和指令存储在单一的 “读／写存储器” 中。

• 存储器的内容通过位置寻址， 而不关心存储在其中的数据类型。

• 以顺序的形式从一条指令到下一条指令的（除非有明确的修改）执行。

形成这些概念的原因在第2:i;t中已经讨论过 ， 但在这里仍值得总结一下。 可以将一小组基本的逻辑部件以各种方式组合起来， 用以存储二进制数据和完成对数据的算术和逻辑操作。 如果要执行一种特定的计算， 需要构造一个专门用于特殊计算的逻辑单元的配肾。 将各种元件连接成所需配置的过程， 可以看成是某种形式的编程。 得到的 “程序” 以硬件的形式存在． 并被称为硬布线程序。

现在考虑另一种方案。 假设我们构造一个具有算术和逻辑功能的通用结构。 这组硬件将根据提供给它的控制信号，对数据执行各种功能。 在原先专用化硬件的情况中， 系统接收数据并生成输出（如图3-1a所示） 而对于通用的硬件， 系统接收数据和控制信号并生成输出。因此，对于每个新程序，程序员只需提供一个新的控制信号集，而不用重新连接硬件。

如何提供控制信号？答案简单又微妙。 整个程序实际上由许多步骤组成． 对某些数据执行某种算术或逻辑操作，每一步都需要一组新的控制信号。让我们为每一组控制信号提供一个唯一的代码，并为通用硬件增加能够接收代码和产生控制信号的部分（如图3-1b所示）。

现在编程容易多了。不再需要为每个新的程序重新连接硬件，所需要做的只是提供新的代码序列。事实上，每个代码就是一条指令，而硬件的一部分翻译每条指令并且产生相应的控制信号。为了区分这一新的编程方法，这一代码或指令序列被称为软件。

图3-1b指出了系统的两个主要部件：指令解释器和通用算数逻辑功能模块。这两部分组成了CPU。为了制造能够工作的计算机还需要其他几个部件。数据和指令必须能够输入系统，为此，需要某种输入模块。这个模块包含几个基本部件，他们能够以某种形式接收数据和指令，并将其转换成系统能够使用的信号的内部形式。需要有某种报告结果的方法，它可以用输出模块的形式实现。两者放到一起，被称为I/O部件。

还需要另外一个部件。输入设备顺序地输入指令和数据，但一个程序并不会始终顺序地执行，它可能跳转到其他地方（例如IAS的跳转指令）。类似地，对数据的操作可能要求不只是以一种预先确定的序列每次访问一个单元。因此，必须有一个可以临时存放指令和数据的地方，这个模块被称为存储器或主存，这种称呼是为了将它同外部存储器或外部设备区分开。冯.诺伊曼指出，同一存储器既可以存放指令又可以存放数据。

图3.2表示了这些顶层部件并暗示了他们之间的相互作用。CPU负责与存储器间交换数据，为了这个目的，CPU一般使用两个内部的寄存器：一个是存储器地址存储器（MAR），为下一次读或写指定存储器的地址；另一个是存储器缓冲寄存器（MBR），容纳写到内存或从内存接收的数据。类似地，I/O地址寄存器（I/O AR）指定了一个特定的I/O设备；I/O缓冲寄存器（I/O BR）用于I/O模块与CPU之间的数据交换。

存储器模块包含一组单元，由连续的编号来定义其地址。每个单元都含有一个二进制数，它既可以解释为指令也可以解释为数据。 I/O模块将数据从外设传送到CPU或存储器， 反之亦然。 I/O模块包含内存缓冲器，用来暂时存放I/O数据，直到它们被发送出去。

以上简单介绍了这些部件，下面将概述这些部件如何共同工作来执行程序的。

3.2 计算机的功能

计算机完成的基本功能是执行程序，该程序由存储在存储器中的一串指令组成。 处理器通过执行程序中指定的指令来完成实际的工作。 本节提供了执行程序的关键元素的概况。在其最简单的形式中， 指令的处理由两个步骤组成：处理器从存储器中每次读取(fetch)一条指令，然后执行每条指令。 程序的执行便是重复地取指令和执行指令的过程。当然， 根据指令的特点，指令的执行可能包含许多步骤（例如 ， 图2.4的下半部）。

一条指令所要求的处理过程被称为指令周期。 根据以上描述的两个简单步骤，图3.3描绘了指令周期的处理步骤。 这两个步骤分别称为取指周期和执行周期。只有当关机或某种不可恢复的错误发生或计算机遇到的是一条停机指令时，程序的执行才停止。

指令的读取和执行

在每个指令周期的开始， 处理器都从存储器中取指令。 在典型的处理器中， 用一个称为程序计数器(PC)的寄存器来保存下一条将要读取指令的地址。 除非特别说明 ， 否则处理器在每次取指令之后总是将PC的值加上一个增量， 以便将来顺序地读取下一条指令（也就是位于下一个更高存储器地址中的指令） 。举例来说， 考虑一台计算机， 其每一条指令占住存储器的一个16位的字单元。 现假设程序计数器的值为存储器位置300, 其中位置地址是指16位字。则处理器下一次将读取300单元中的指令； 再下一个指令周期． 它将取301单元中的指令， 接着是302、 303等。 正如刚才所解释的， 这一顺序允许有所改变。

读取的指令装入处理器中的指令寄存器(IR)。 指令以二进制代码的形式存在， 它规定了处理器将要执行的动作。 处理器解释这条指令井执行所要求的操作。 总的来说，这些橾作归为4类：

• 处理器－存储器：数据可从处理器传送到存储器或从存储器传送到处理器。

• 处理器-I/O： 通过处理器和I/O模块之间的传输， 数据可传送到或来自外部设备。

• 数据处理：处理器可以对数据执行一些算术或逻辑操作。

• 控制：指令可以用来改变执行顺序。例如， 处理器可能从149单元取得一条指令 而这条指令指出下一条指令取自182单元。 处理器通过将程序计数器设置为182来记录这一事实。这样，在下一个取指令周期， 指令将取自182单元， 而不是150单元。

当然，一条指令的执行可能包含这些操作的组合。

考虑一个简单的例子， 使用一台包含图3.4所列特点的假想机器， 其处理器包含唯一的一个数据寄存器， 被称为累加器(AC); 其指令和数据都是16位长， 这样便于用16位的字来组织存储器：其指令格式提供4位的操作码， 表示最多可以有2^4=16种不同的操作码， 最多有2^12= 4096(4K}个字的存储器可以直接寻址。

图3.5举例说明部分程序的执行，显示了存储器和处理器寄存器的相关部分。该程序段将存储器中地址940的内容与地址941的内容相加，结果放在941中。这里需要3条指令，他们可用3个指令周期和3个执行周期来表示：

1. 程序计数器（PC）的内容是300，即第1条指令的地址。这条指令（其值为十六进制数1940）被装入指令寄存器IR, 并且PC加1。 注意，这一过程包含对存储器地址寄存器和存储器缓冲寄存器的使用。为简便起见，忽略了这些中间寄存器。
2. IR中的前4位（第1个十六进制数字）指出要装入累加器(AC), 而其余12位(3个十六进制数字）指定从那个地址(940)取数据装载。 即地址940的数据0003装入AC。
3. 从单元301中取下一条指令（5941），并且PC加1。
4. AC中存放的内容和941单元的内容相加， 结果放入AC。
5. 从单元302中取下一条指令(2941), 并且PC加1。
6. 将AC的内容存人941单元。

在这个例子中，将940单元的内容加到941单元用了3个指令周期，每个指令周期都包含了一个取指周期和一个执行周期。如果用更复杂的指令集， 则需要更少的周期。 例如 ， 一些较老的处理器包含具有多个存储器地址的指令 ， 因此 ， 这类处理器中的特殊指令在执行周期可以多次访问存储器。 而且 ， 指令可能不用访问存储器 ， 而是指定一个I/O操作。

例如， PDP-11处理器包含一条指令， 其符号表示为"ADD B,A", 它将存储器单元A和B 相加 ， 然后将和存入单元A中。 这一个指令周期由下列几步组成：

• 取ADD指令。

• 将存储单元A的内容读入处理器。

• 将存储单元B的内容读入处理器。 为了使A的内容不丢失，处理器必须至少有两个寄存器 而不是一个单一的累加器来存放存储器的值。

• 将两个值相加。

• 将结果从处理器写入内存单元A中。

因此，某个特定指令的执行周期可能包含对存储器的多次访问。而且，除访问存储器以外，一条指令可能指定一次I/O操作。 考虑到这个附加的因素，图3.6对图3.3的基本指令周期提供了更详细的考虑，它以状态图的形式显示。 对于任意给定的指令周期，有些状态可能为空，而另一些可能出现多次。 这些状态的描述如下所示：

• 指令地址计算（iac）, 决定下一条将要执行的指令的地址。 通常是将一个固定的值与前一条指令的地址相加。 例如， 如果每条指令有16位长， 并且存储器是由16位字构成的， 则将原地址加1; 如果存储器是由可独立寻址的8位字节构成的， 则将原地址加2。

• 读取指令(if) 将指令从存储器单元读到处理器中。

• 指令操作译码（iod）： 分析指令， 以决定将执行何种操作以及将使用的操作数。

• 操作数地址计算(oac)： 如果该操作包含对存储器或通过I/O的操作数访问， 那么决定操作数的地址。

• 取操作数(of)：从存储器或从I/O中读取操作数。

• 数据操作(do)：完成指令需要的操作。

• 存储操作数(os)：将结果写人存储器或输出到I/O。

图3-6上半部分的状态图包含处理器与存储器或I/O模块的数据交换。 图3.6下半部分的状态图仅涉及处理器内部的操作。oac状态出现了两次 ， 这是因为指令包含了读、 写或两者兼而有之。 但在两种情况中该状态所完成的操作基本相同 ， 因 此只需要一个状态标识。

还请注意， 图3.6允许多个操作数和多个结果， 因为指令有些和有些机器要求这样。 例如， PDP-11的"ADD A,B" 指令导致以下的状态序列：iac, if, iod, oac, of, oac, of, do, oac, os。

最后 ， 在某些机器中，单条指令可以指定对向量 (一维数组）数据或字符串（一维数组） 执行操作。 如图3.6所示， 它将重复取操作数和（或）存储操作数。

中断

几乎所有的计算机都提供一种机制 ， 其他模块(I/O或存储器）通过此机制可以中断处理器的正常处理。 表3.1列出了最常见的中断类型。 这些中断的特点将在本书的后半部分， 特别是在 第7章和第14章中予以讨论。 但是为了更清楚地理解指令周期的特性和中断对互连结构的影响 ， 有必要在此介绍这个概念。 读者在本阶段不用关心中断的产生和处理等细节 ， 只需专注由中断引起的没模块之间的通信。

提供中断主要是为了提高处理的效率。例如，大部分外设比处理器慢很多。假如处理器使用如图3.3所示的指令周期的方法将数据传送给打印机，在每次写操作之后，处理器都会暂停并处于空闲状态，直到打印机跟上进度。暂停的时间可能有几百个甚至上千个指令周期，这还不包括存储器。显然，这是处理器使用上的巨大浪费。

图3-7a说明了事件的这种状态。用户程序执行一系列WRITE调用，WRITE调用与处理过程交错进行。代码段1、2、3是指不包含I/O操作的指令序列。WRITE调用是对一段I/O程序的调用，它是执行实际I/O操作的系统实用程序。这段I/O程序包含以下三个部分：

• 用于为实际I/O操作准备的指令序列，在图中标记为4。它可能包括将待输出数据复制到专用的缓冲区，以及为设备命令准备参数。

• 实际的I/O命令。如果没有中断的使用，一旦此命令发出，程序必须等待I/O设备完成需要的功能（或周期性地测试设备）。程序可以通过简单地重复执行一个测试操作来决定该I/O操作是否完成。

• 完成该操作的指令序列，在图中标记为5。这可能包含设置标志位来表示操作是成功还是失败

因为I/O操作可能需要花较长的时间才能完成，所以I/O程序挂起，等待操作完成；于是， 用户程序在WRITE调用这一点暂停很长一段时间。

中断和指令周期

有了中断，处理器就可以在I/O操作进行的时候执行其他指令。考虑如图3.7b所示的控制流。与前面相同，用户程序到达这一点，以WRITE调用的形式进行系统调用。此时，被调用的I/O程序仅仅由准备代码和实际的I/O命令组成。当这几条指令执行后，控制权返回给用户程序。同时外部设备忙于从计算机存储器中接收数据并打印。这次I/O操作与用户程序中的指令执行同时进行。

当外部设备准备好接收服务，即当它准备从处理器中接受更多的数据时，外部设备的I/O模块发送中断请求信号给处理器。处理器通过挂起当前程序的操作，跳转服务于某个特定I/O设备的程序来响应，这个程序被称为中断处理程序，并且在设备服务完后恢复原来的执行。中断发生的断电在图3.7a中用星号（\*）表示。

我们来解释一下图3.7表示的内容。有一个包含两个WRITE命令的用户程序。 在开始处有一段代码，然后是一个WRITE命令，之后是第二段代码和第二个WRITE命令，然后是第三个也是最后一段代码。 WRITE命令调用OS提供的I / O程序。 类似地，I / O程序由一段代码组成，然后是I / O命令，后面跟着另一段代码.I / O命令调用硬件I / O操作。

从用户程序的角度来看，中断只是打断正常的执行序列。当中断处理完成之后，恢复执行原来的序列（如图3.8所示）。因此，用户程序不必为适应中断而提供任何特殊代码。处理器和操作系统负责用户程序挂起和在中断点处恢复等操作。

为适应中断，将中断周期加入指令周期中，如图3.9所示。在中断周期中，处理器检查是否发生了中断，这将由中断请求信号的出现来指示。如果没有中断请求，则处理器继续进入取指周期，读取当前程序的下一条指令。如果出现中断请求，则处理器执行以下操作：

挂起当前正在执行的程序，并保存其状态。这意味着保存下一条即将执行的指令的地址（程序计数器PC的当前内容）以及任何与处理器当前活动相关的数据。

将程序计数器设置为中断处理程序的起始地址。

处理器现在进入取指周期，并读取服务于该中断的中断处理程序的第1条指令。中断处理程序一般是操作系统的一部分。通常中断处理程序判定中断的性质并且完成所需要的任何操作。在我们已经使用过的例子中，中断处理程序判断是哪个I/O模块产生了中断，然后跳转到向I/O模块写数据的程序。中断处理程序结束后，处理器能够在断点处恢复用户程序的执行。

显然，这一处理增加了系统的开销。必须执行额外的指令（在中断处理程序中）来判断中断的性质并决定相应的操作。然而，由于简单地等待I/O操作要浪费大量的时间，因此中断的使用能使处理器更有效地运行。

为了评价获得的效率增益，考虑图3.10，它是基于图3.7a和图3.7b控制流的时序简图。 在此图中，用户程序代码段呈绿色阴影，而I/O程序代码段呈灰色阴影。 图3.10a显示了不使用中断的情况。当执行I/O操作时，处理器必须等待。

图3.7b和图3.10假设I/O操作所需的时间较短：比完成用户程序中写操作之间的指令执行时间要短。在这种情况下，代码段2被中断。 执行一部分代码（2a）（执行I / O操作时），然后发生中断（I / O操作完成时）。 在中断服务之后，将继续执行代码段2（2b）的其余部分。

更典型的情况，对于打印机之类的慢速设备，I/O操作的时间要比执行一系列用户指令的时间长很多，图3.7c给出了这种情况的状况。在这种情况下，用户程序在第1次WRITE调用产生的I/O操作完成之前，就面临第2次WRITE调用，结果是用户程序在这一点挂起。当前一个I/O操作完成之后，才可能处理新的WRITE调用，启动新的I/O操作。图3.11显示了这种情况下使用中断和未使用中断的时序。无论从哪种情况下都可以看出，效率得到了提高，因为部分I/O操作的时间与用户指令执行的时间相重叠。

图3.12显示了修改后包含中断周期处理的指令周期状态图。

多重中断

迄今为止， 只讨论了发生单个中断的情形。但是实际的情况是可能产生多个中断。 例如 ， 一个程序可以从通估线路接收数据和打印结果。 打印机每完成一次打印操作便会产生一次中断。 每次到达一个单元的数据， 通信线路控制器将产生一次中断。 根据通信规程， 一个单元可以是一个字符 ， 也可以是一个数据块。 任何情况下． 在打印机中断处理时都可能再发生通信中断。

处理多礼中断有两种方法。第1种是在中断处理过程中禁止其他中断。 禁止中断仅仅惹味籽处理器可以井且将忽略中断请求信号。 如果中断在此时发生， 一般会保持在 “ 未决状态” ． 在处理器允许中断后就会检测到这种未决状态。 于是， 当用户程序执行时如果有一个中断发生， 则该中断会立即被禁止。在中断处理程序完成后，不用等到用户程序恢复就可以再次允许中断，这时候处理器检查是否发生了其他中断。这种方法既简单又有效，因为中断严格桉顺序处理（如图3.13所示）。

上面这种方法的缺点是没有考虑到相对的优先级和时间紧迫的需要。例如，当输人数据从通信线路到达时，需要被迅速接收，以便给更多的输人腾出空间。如果第2批数据到达之前第1 批数据还没有处理，就可能丢失数据。

第2种方法是定义中断的优先级，且允许优先级高的中断引起低级中断处理程序本身被中断 （如图3-!3b所示）。作为第2种方法的例子，考虑一个有3个110设备的系统打印机、硬盘和通信线路，它们的优先级逐个递!1!,分别是2、4、5。图3-14举出了一个可能的序列。用户程序开始于'=0时刻。当'=10时，发生了打印机中断，用户信息放入系统栈，并继续从打印机的中断服务程序(JSR)开始执行。当程序仍在执行时，在,= 15 时刻，通信中断发生。由于通信线的优先级比打印机高，这个中断钳到响应。打印机JSR被中 断它的状态压人栈，继续从通信JSR执行。当这个通信JSR正在执行时，发生了磁盘中断(t=20)由于它的优先级相对较低，只好挂起，而通信JSR运行到结束。

当通信ISR完成时（t=25），原来的处理器的状态恢复，即执行原来的打印机ISR。 但处在这一例程中的一条指令都没打来倡及执行以前， 处理器响应优先级更庙的磁盘中断， 将控制权传送给磁盘JSR。 仅当这一例程结束后(t=35), 打印机的JSR才恢复。 在它完成后(t=40) 控制权才最终交还给用户程序。

I/O功能

至此 ． 我们已经讨论了由处理器控制的计算机操作， 而且主要考察了处理器和存储器的交 互橾作。 前面的讨论只提到了VO部件的作用． 这一作用将在第7东中详细讨论， 但这里有必要给出一个简单的介绍。

VO校块（例如磁盘控制器）能直接与处理器交换数据。 如同处理器通过指定某个单元地址就可以启动一次对存估器的读／写一样， 处理器同样也能把数据写到VO模块或从VO校块中读出。 在后一种情况中， 处理器识别由特定VO模块控制的特定设备。 因此， 会发生与图3.5中形式相似的指令序列 ， 只不过它们是VO指令 ， 而不是存储器访问指令。

在某些情况下 ， 需要允许VO直接与存储器交换数据。 在这种情况中， 处理器授予VO模块读或写存储器的权利 ． 以便VO和存储器的传输不需要CPU的介入。 在这种传输中， VO模块向存储器发出读或写的命令 ， 解脱了处理器负责数据交换的责任。 这种操作称为宜接存储器访问 (OMA), 将在第7兹中详细论述。

3.3 互连结构

计n:机包含一组部件或3种基本类型的校块（处理器、 存忧器和VO), 模块之间相互通信。 实际上， 计算机足一个基本校块的网络， 因此必须有连接这些栈块的通路。

连桵各种模块的通路的梨合称为互连结构(l,tereo,""""'structore) 。 这种结构的设计将取决于模块之间所必须进行的交换。

图3-15通过指出每种栈块类型的主要输人 、 输出形式给出了所需的信息交换的种类：

存储器：通常 ， 存储器模块由N个等长的字组成， 每个字分配了一个唯一的数值地址(0、1、...、N-1)，数据字可以从存储器中读出或写进存储器。操作的性质由读和写控制信号指示，操作的单元由地址指定。

I/O模块：从（计算机系统）内部的观点乔， I/O在功能上与存储器相似， 它们都有读和写两类操作。 此外，一个I/O模块可以控制多个外设。 我们可以定义每个与外部设备的接口为端口(port)，给它分配一个唯一的地址（例如， 0 、 I, M-1)。 此外 ， 还有向外部设备输人和输小数据的外部数据路径。最后，I/O模块可以给处理器发送中断信号。

处理器：处理器读入指令和数据，并在处理之后写出数据，它还用控制信号控制整个系统的操作。也可以接收中断信号。

前面所列定义了要交换的数据。互连结构必须支持下列类型的传送：

存储器到处理器：处理器从存忧器中读一条指令或一个单元的数据。

处理器到存储器：处理器向存储器写一个单元的数据。

I/O到处理器：处理器通过I/O模块从I/O设备中读数据。

处理器到I/O：处理器向I/O设备发送数据。

I/O与存储器之间：对于这两种情况，I/O模块允许与存储器直接交换数据，使用直接存储器存储，而不通过处理器。

多年来，人们尝试过各种各样的互连结构。目前最常见的是(1)总线和各种多总线结构，以及(2)具有分组数据传输的点对点互连结构。本章后续部分将专门讨论这些结构。

3.4总线互连

几十年来，总线一直是计算机系统部件互连的主要方式。对于一般的计算机，它逐渐被各种各样的点对点互连结构所取代，而这些结构现在主导着计算机系统的设计。然而，总线结构仍然普遍用于嵌入式系统，特别是微控制器。本节简要介绍总线结构。更详细的内容参见附录。

总线(611,)是忐接两个或多个设备的通信通路。 总线的关键特征是共享传输介质。 多个设备连接到总线上 ， 并且任何一个设备发出的信号可以被其他所有连接到总线L的设备所接收。 如果两个设备同时发送， 它们的信号将会赁叠， 这样会引起混沿。 因此 ， 每次只能有一个设备能够成功地利用总线发送数据。

通常，总线由多条通信路径或线路组成，每条线能够传送代表二进制1和0的信号。一段时间里，一条线能传送一串二进制数字。总线的几条线放在一起，就能够用来同时（并行地）传送二进制数字。例如，一个8位的数据能通过总线中的8条线传送。

计算机系统含有多种总线 ， 它们在计算机系统的各个层次提供部件之间的通路。 连接计算 机的主要部件（处理器 、 存储器 、 I/O)的总线称为系统总线。 最常见的计算机互连结构是基于一个或多个系统总线的使用。

系统总线通常包含so到上百条分立的导线， 每条导线被赋予一个特定的含义或功能。 虽然总线的设计有多种 ， 但任何总线的线路都可以分成如下3个功能组（如图3-16所示） 数据线 ， 地址线和控制线。 此外 ， 还有为连接的校块提供屯游的电源馈线。

数据线提供系统校块间传送数据的路径， 这些线组合在一起称为数据总线。 典型的数据总线包含32 、 64 、 128或更多的分离导线， 这些线的数目称为数据总线的宽度。 因为每条线每次能传送1位，所以线的数目决定了每次能同时传送多少位 ， 数据总线宽度是决定系统总体性能的关键因索。 例如， 如果数拼总线为32位宽， 而每条指令 64位长 ． 那么处理器在每个指令周期必须访问存忧器校块两次。

地址线用千指定数据总线上数据的米说或去向。 例如 ， 如果处理器希型从存估器中读取一 个字(8位 、 16位或32位） ， 它将所符要的字的地址放在地址线L。 显然 ， 地址总线的宽度决定了系统能够使用的蚊大的存估器容从。 而且， 地址线通常也用于VO瑞口的寻址。 通甘， 地址线的岛位用于选择总线上指定的校块， 低位用于选择栈块内具体的存忧器单元或 1/0 端口 。 例 如 在一个8位地址总线上， 小千等于01111111的地址可以用来访问有 128个字的存储器模块（模块0），而大于等于10000000的地址可用来访问接在I/O模块上的设备（模块1）。

控制线用来控制对数据线和地址线的存取和使用。 由于数据线和地址线被所有校块共享 ， 因此必须用一种方法来控制它们的使用。 控制信号在系统校块之间发送命令和时序信号。 时序 信号指定了数据和地址信号的打效性． 命令信号指定了要执行的操作。 典型的控制信号如下所列：

• 存储器写(Memory Wn<e) 引起总线上的数据写人被寻址的单元。

• 存储器读(Memory Read) 使所寻址单元的数据放到总线上。

• 1/0写 (l/0 Wn<e), 引起总线上的数据输出到被寻址的 1/0 端口 。

• 1/0读 (1/0 R邸d) 使被寻址的 1/0 端口的数据放到总线上。

• 传输响应(Traa,f«ACK), 表示数据巳经从总线上接收． 或已经将数据放到总线上。

• 总线请求(B,s R叩闷） 表示校块需要获得对总线的控制。

• 总线允许(B"'Gran<), 表示发出诮求的校块已经被允许控割总线。

• 中断请求(Interrupt Request） 表示某个中断正在悬而未决。

• 中断响应(Interrupt ACK), 未决的中断请求被响应。

• 时钟(Clock） 用于同步操作。

• 复位(Reset), 初始化所有模块。

总线的操作如下，如果一个模块希望向另一个模块发送数据，它必须做两件事情：（1）获得总线的使用权；（2）通过总线传送数据。如果一个模块希望向另一个模块请求数据，它也必须做两件事情：（1）获得总线的使用权；（2）通过适当的地址线和控制线向另一模块发送请求，然后它必须等待另一模块发送数据。

3.5点对点互连

几十年来，共享总线架构是处理器和其他组件(内存、I/O等)之间互连的标准方法。但现代系统越来越依赖点对点的互连，而不是共享总线。

驱动从总线到点对点互连的变化，主要原因是随着宽同步总线频率的增加而遇到的电力限制。数据速率越来越高，要及时地执行同步和仲裁功能变得越来越困难。此外，随着多核芯片的出现，在一个芯片上有多个处理器和大量内存，人们发现在同一芯片上使用传统的共享总线增加了提高总线数据速率和减少总线延迟以跟上处理器的速度的困难。与共享总线相比，点对点互连具有更低的延迟、更高的数据速率和更好的可扩展性。

在本节中，我们将看到点对点互连方法的一个重要且具有代表性的示例:2008年引入的Intel的QuickPath互连(QPI)。

以下是QPI和其他点对点互连方案的重要特征：

多重直接连接：系统内的多个组件可直接与其他组件进行成对连接。这消除了在共享传输系统中发现仲裁的需要。

分层协议架构：正如网络环境（如基于TCP / IP的数据网络）中所发现的，这些处理器级互连使用分层协议架构，而不是简单使用共享总线配置中的控制信号。

分组数据传输：数据不作为原始比特流发送。 相反，数据是作为一系列数据包发送的，每个数据包都包含控制标题和错误控制代码。

图3.17说明了在多核计算机上QPI的典型用法。QPI链接（由图中的绿色箭头对表示）形成一个交换结构，使数据能够在整个网络中移动。可以在每对核心处理器之间建立直接QPI连接。如果图3.17中的内核A需要访问内核D中的存储器控制器，那么它会通过内核B或C发送请求，而B或C必须依次将该请求转发到内核D中的内存控制器。类似地，可以使用具有三个链接的处理器并通过中间处理器路由通信来构建具有八个或多个处理器的大型系统。此外，QPI用于连接一个I/O模块，称为I/O hub (IOH)。IOH充当交换机，用于在I/O设备之间进行通信。通常在较新的系统中，从IOH到I/O设备控制器的链路通常使用称为PCI Express（PCIe）的互连技术，本章后面将对此进行描述。IOH在QPI协议和格式以及PCIe协议和格式之间进行转换。内核还使用专用内存总线连接到主内存模块(通常内存使用动态访问随机内存(DRAM)技术)。

QPI被定义为四层协议体系结构，包含以下层（图3.18）：

物理：由传输信号的实际导线，以及支持传输和接收1和0所需辅助功能的电路和逻辑组成。 物理层的传输单位是20位，被称为Phit(物理单元)。

链接：负责可靠的传输和流量控制。 链路层的传输单元是一个80位的Flit（流量控制单元）。

路由：提供了通过结构引导数据包的框架。

协议：设备之间交换数据包的高级规则集。 数据包由整数个Flits组成。

QPI物理层

图3.19显示了QPI端口的物理体系结构。 QPI端口由84个独立的链路组成，如下所示。 每个数据路径由一对一次传输数据的导线组成; 这一对被称为一条通道。每个方向有20条数据通道（发送和接收），以及每个方向的时钟通道。 因此，QPI能够在每个方向并行传输20位。 20位单元被称为phit。 目前产品链路的典型信令速度要求以6.4GT / s（每秒传输）的速度运行。 每传输20位，总计达16 GB / s，并且由于QPI链接涉及专用双向对，因此总容量为32 GB / s。

每个方向的车道分为四个象限，每个象限有五个通道。在某些应用中，链路也可以在一半或四分之一宽度下运行，以降低功耗或解决故障。

每条通道上的传输形式称为差分信号传输或均衡传输。采用均衡传输时，信号沿着一个导体作为电流传输，然后返回另一个导体。二进制值取决于电压差。通常，一条线具有正电压值，另一条线具有零电压，一条线与二进制1关联，一条线与二进制0关联。具体而言，QPI使用的技术被称为低电压差分信号（LVDS ）。在一般的实现中，发射器根据要发送的逻辑电平将一个小电流注入到其中一条线中。电流通过接收端上的一个电阻，沿着另一根导线往相反的方向返回。接收器检测电阻两端的电压极性来确定逻辑电平。

物理层执行的另一个功能是，它使用一种称为多通道分布的技术来管理80位FITS和20位PHIT之间的转换。如图3.20所示，数据流可以看作以循环方式分布在数据通道上的比特流（第一比特到第一通道，第二比特到第二通道等）。这种方法使得QPI能够通过实现两个端口之间的物理链路作为多个并行信道来实现非常高的数据速率。

QPI链路层

QPI链路层有两个关键功能：流量控制和差错控制。这些功能是作为QPI链路层协议的一部分执行的，并且在微处理器（流量控制单元）的级别上进行操作。每个微片包含一个72位的消息有效载荷和一个称为循环冗余校验（CRC）的8位错误控制代码。我们将在第5章讨论错误控制代码。

一个flit有效载荷可以包含数据或消息信息。数据流在内核之间或内核与IOH之间传输实际的数据位。消息flits用于流量控制、差错控制和缓存一致性等功能。我们将在第5章和第17章讨论缓存一致性。

流量控制功能是必需的，以确保发送QPI实体不会以比接收方处理数据更快的速度发送数据，并为更多的传入数据清除缓冲区。为了控制数据流，QPI使用信用计划。在初始化过程中，发送者有一定数量的窗口值以发送到接收者内容。无论何时发送给接收方，发送方都会将其窗口值减1。无论何时接收方释放缓冲区之后，都会向该缓冲区的发送方返回一个释放的窗口数目。因此，接收器控制数据通过QPI链路传输的速度。

有时由于噪声或其他现象，传输过程中在物理层传输的比特会发生变化。链路层的差错控制功能检测并从这些位错误中恢复，从而使得较高层不会发生这种错误。对于从系统A到系统B的数据流，过程如下所示：

1.如前所述，每个80位的微处理器都包含一个8位的CRC字段。 CRC是其余72位值的函数。 在传输时，A计算每个微片的CRC值并将该值插入微片。

2.当收到一个微片时，B计算72位有效负载的CRC值，并将该值与微片中的传入CRC值进行比较。 如果两个CRC值不匹配，则检测到错误。

3.当B检测到错误时，它向A发送请求以重新发送错误的微片。 但是，因为A可能有足够的窗口来发送一个短暂的流，所以在错误超时之后并且在A收到重新发送的请求之前已经发送了其他消息。 因此，请求是A备份并重新传输损坏的微件加上所有后续的微件。

QPI路由层

路由层用于确定数据包在整个可用系统互连中的走向。路由表由固件定义并记录数据包可能遵循的路径。在小型配置中，例如双路平台，路由选项是有限的，而路由表非常简单。对于较大的系统，路由表选项更加复杂，可根据平台中如何（1）设备填充，（2）对系统资源进行分区，以及（3）可靠性事件导致在一个失败的资源周围映射，从而提供路由和重路由流量的灵活性。

QPI协议层

在这一层，数据包被定义为传输单位。数据包内容的定义是标准化的，具有一定的灵活性，可以满足不同的市场需求。在此级别执行的关键功能是高速缓存一致性协议，该协议处理确保多个高速缓存中保存的主内存值一致。典型的数据包有效载荷是发送到缓存或从缓存发送的数据块。

3.6 PCI Express

外设部件互连（PCI）总线是一种高带宽、独立于处理器的总线，它能够作为中间层或外围设备总线。与其他普通的总线规范相比 ， PCI为高速的I/O子系统（例如 ， 图形显示适配器、 网络接口控制器 、 磁盘控制器）提供了更好的性能。

Intel在1990年开始为其Peatmm系统开发PC!。 很快 ， '"'"将所有的专利向外界公开， 井促进了工业协会， PC! SIG (PC/特别兴趣组）的创建， 它的任务是进一步开发并维护PC! 规范的兼容性。 结果是PC! 被广泛地采纳 ， 越来越多地应用到个人计算机 ， 工作站以及服务器系统中。 巾于这个规范是公开的， 而且它得到了许多改处理器和外围设备生产商的支持， 因此不同生产商的PC! 产品是相互兼容的。

与前面讨论的系统总线一样，基于总线的PCI方案也未能跟上连接设备的数据速率需求。因此，开发了一种称为PCI Express（PCIe）的新版本。与QPI一样，PCIe是一种旨在替代基于总线的方案（如PCI）的点对点互连方案。

PCIe的关键在于高容量，以支持更高数据速率I / O设备（如千兆以太网）的需求。另一个需求涉及支持依赖时间的数据流的需要。视频点播和音频重分发等应用程序也对服务器施加了实时限制。许多通信应用程序和嵌入式PC控制系统也实时处理数据。如今的平台还必须处理不断增长的数据速率下的多个并发传输。不再接受将所有数据都视为平等的——例如，首先处理流数据更为重要，因为后期实时数据与没有数据一样无用。需要对数据进行标记，以便I / O系统可以优先考虑整个平台的流。

PCI物理与逻辑体系结构

图3.21是支持使用PCIE的典型配置。根联合体设备(也称为芯片组或主机桥)将处理器和内存子系统连接到由一个或多个PCIe和PCIe交换机设备组成的PCI Express交换机结构。根联合体充当缓冲设备，用于处理I / O控制器与内存和处理器组件之间的数据速率差异。根联合体也在PCIe事务格式和处理器、内存信号和控制需求之间进行转换。芯片组通常会支持多个PCIe端口，其中一些端口直接连接到PCIe设备，另外一个或多个连接到管理多个PCIe流的交换机。来自芯片组的PCIe链接可能会连接到下列实现PCIe的设备：

交换机：交换机管理多个PCIe流。

PCIe端点：实现PCIe的I / O设备或控制器，例如千兆以太网交换机，图形或视频控制器，磁盘接口或通信控制器。

遗留端点：遗留端点类别适用于已迁移到PCI Express的现有设计，它允许使用I/O空间和锁定事务等遗留行为。PCI Express端点不允许在运行时使用I/O空间，不能使用锁定事务。通过区分这些类别，系统设计者有可能限制或消除对系统性能和鲁棒性有负面影响的遗留行为。

PCIe / PCI桥：允许将较早的PCI设备连接到基于PCIe的系统。与QPI一样，PCIe交互使用协议架构来定义。 PCIe协议体系结构包含以下层（图3.22）：

物理层：由传输信号的实际导线，以及支持传输和接收1和0所需的辅助功能的电路和逻辑组成。

数据链路层：负责可靠的传输和流量控制。由DLL生成和使用的数据包称为数据链路层包（DLLPs）。

事务层：生成和使用用于实现加载/存储数据传输机制的数据包，并管理链路上两个组件之间的数据包流控制。由TL生成和使用的数据包称为事务层包（TLP）。

在TL上方是软件层，它们生成读取和写入请求，这些读取和写入请求由事务层使用基于数据包的事务协议传输到I / O设备。

PCIe物理层

与QPI类似，PCIe是一种点对点体系结构。 每个PCIe端口由多个双向通道组成（请注意，在QPI中，通道仅指单向传输）。 在一个通道中的每个方向传输都是通过一对电线上的差分信号传输。 一个PCI端口可以提供1,4,6,16或32条通道。 在下文中，我们将参考2010年底推出的PCIe 3.0规范。

与QPI一样，PCIe使用多通道分配技术。 图3.23显示了由四条通道组成的PCIe端口示例。 数据通过简单的循环方案一次分配到四个通道1个字节。 在每个物理通道上，一次缓冲并处理16个字节（128位）的数据。 每个128位的块被编码为一个唯一的130位码字进行传输; 这被称为128b / 130b编码。 因此，单个通道的有效数据速率降低了128/130。

要理解128b / 130b编码的基本原理，首先要注意它与QPI不同，PCIe不使用其时钟线来同步比特流。也就是说，时钟线不用于确定每个输入位的开始和结束点;它仅用于其他用途。但是接收器必须与发送器同步，以便接收器知道每个位何时开始何时结束。如果用于比特传输和接收发射机接收机的时钟之间有转移，就可能发生错误。为了处理发生这种转移的可能性，PCIe依靠基于发射信号的接收机与发射机同步。与QPI一样，PCIe使用一对电线上的差分信号。接收器可以通过寻找数据转换并将其时钟同步到转换来实现同步。但是，考虑到使用差分信号的长串1或0，输出在很长一段时间内是恒定电压。在这种情况下，发射机和接收机的时钟之间的任何转移都会导致两者之间同步失败。

一种常见的方法(PCIe 3.0中使用的方法)是加扰。加扰(不会增加要传输的位元数)是一种映射技术，它会使数据看起来更随机。置乱倾向于分散转换的数量，使它们在接收端出现的间隔更均匀，这有利于同步。此外，如果数据更接近于随机性质，而不是常数或者重复，则会增强其他传输属性，如光谱属性。有关加扰的更多讨论，请参见附录E。

另一种有助于同步的技术是编码，在编码中，将额外的位插入位流以强制转换。对于PCIe 3.0，通过添加2位块同步头，每组128位输入被映射到130位块。数据块的标题值为10，所谓的有序集块为链接级信息块。

图3.24说明了使用加扰和编码技术。 要传输的数据被送入一个扰码器。 然后将加扰输出送入128b / 130b编码器，该编码器缓存128位，然后将128位块映射到130位块。该块经过并行 - 串行转换器并使用差分信令一次传输一位。

在接收器处，时钟与输入数据同步以恢复比特流。 然后通过一个串行到并行转换器产生一个130位的数据流。 每个数据块通过一个128b / 130b解码器，以恢复原始的扰码位模式，然后将其解扰以产生原始位流。

使用这些技术，可以实现16 GB / s的数据速率。 最后提到的一个细节; PCI链路上数据块的每个传输以8位成帧序列开始和结束，旨在使接收器有时间与进入的物理层位流同步。

PCIe事务处理层

事务层（TL）从TL上方的软件接收读取和写入请求，并创建请求数据包以通过链路层传输到目的地。大多数交易使用分割交易技术，按以下方式工作。请求数据包由源PCIe设备发出，然后等待响应，称为完成数据包。在完成请求之后，只有在完成数据或状态准备交付时，完成请求才启动完成。每个分组具有唯一的标识符，使得完成分组被定向到正确的始发者。采用拆分技术时，完成时间与请求时间分开，与典型的总线操作相比，交易双方必须可用来占用和使用总线。在请求和完成之间，其他PCIe流量可以使用该链接。

TL消息和一些写入事务是被发布事务，这意味着没有预期的响应。

TL数据包格式支持32位内存寻址和扩展64位内存寻址。数据包还具有诸如“no -snoop”，“弛豫排序”和“优先级”等属性，这些属性可用于在I/O子系统中优先路由这些信息包。

地址空间和事务类型TL支持四类地址空间：

内存：内存空间包含系统主内存。它还包括PCIe I / O设备。某些范围的内存地址映射到I / O设备。

I / O：该地址空间用于传统PCI设备，保留的内存地址范围用于处理遗留的I/O设备。配置：该地址空间使TL能够读/写配置。

消息：该地址空间用于与中断，错误处理和电源管理相关的控制信号。

表3.2显示了TL提供的事物类型。对于内存，I / O和配置地址空间，有读写事务。在内存事务的情况下，还有一个读锁定请求函数。锁定操作是由于设备驱动程序请求对PCIe设备上的寄存器进行原子访问的结果。例如，设备驱动程序可以自动读取，修改，然后写入设备寄存器。为了实现这一点，设备驱动程序让处理器执行一条指令或一组指令。根联合体将这些处理器指令转换为一系列PCIe事务，这些事务对设备驱动程序执行单独的读取和写入请求。如果这些事务必须以原子方式执行，则根联合会在执行事务时锁定PCIe链路。此锁定可防止不属于序列的事务发生。这一系列事务称为锁定操作。可能导致锁定操作发生的特定处理器指令集取决于系统芯片组和处理器架构。

为了保持与PCI的兼容性，PCIe支持类型0和类型1配置周期。类型1周期向下游传播，直到它到达承载目标设备所在总线（链路）的桥接器接口。配置事务在目标链路上由桥接器从类型1转换为类型0。

最后，完成消息与拆分事务一起使用，用于内存、I/O和配置事务。

tlp数据包组件PCIe事务使用事务层数据包传送，如图3.25a所示。 TLP始于从设备的事务层发送，并终止于接收设备的事务处理层。

上层软件向TL发送TL创建TLP内核所需的信息，其中包含以下字段：

头文件：头文件描述了数据包的类型，包括接收方处理数据包所需的信息，包括任何所需的路由信息。 内部头文件格式将在后面讨论。

数据：TLP中最多包含4096字节的数据字段。 一些TLP不包含数据字段。

ECRC：可选的端到端CRC字段使目标TL层能够检查TLP报头和数据部分中的错误。

PCIe数据链路层

PCIe数据链路层的目的是确保PCIe链路上的数据包可靠传输。 该DLL参与TLP的形成并且还传输DLLP。

数据链路层分组数据链路层分组始发于发送设备的数据链路层并终止于链路另一端设备的DLL。 图3.25b显示了DLLP的格式。 有三个重要的DLLP组用于管理链路：流量控制数据包，电源管理数据包和TLP ACK与NAK数据包。 电源管理数据包用于管理电源平台预算。 流量控制分组调节TLP和DLLP在链路上传输的速率。 ACK和NAK数据包用于TLP处理，下面将会讨论。

事务层数据包处理

该DLL向TL（图3.25a）创建的TLP的核心添加了两个字段：一个16位序列号和一个32位链路层CRC（LCRC）。而在TL中创建的核心字段仅用于目标TL，由DLL添加的两个字段在从源到目的地的每个中间节点上处理。

当TLP到达设备时，DLL将剥离序列号和LCRC字段并检查LCRC。有两种可能的结果：

1.如果未检测到错误，则将TLP的内核部分交给本地事务处理层。如果这个接收设备是预期的目的地，那么TL处理TLP。否则，TL为TLP确定一条路由，并将其传递回DLL，继续通过下一个链路进行传输。

2.如果检测到错误，则DLL调度NAK DLL数据包以返回到远程发射机。 淘汰TLP。

当DLL传输TLP时，它保留TLP的副本。如果收到TLP的NAK，则重新发送TLP。当它收到ACK时，则丢弃缓冲的TLP。

3.7 关键词、思考题和习题

关键词

address bus地址总线

address lines地址线

arbitration仲裁

balanced transmission平衡传输

bus总线

control lines控制线

data bus数据总线

data lines数据线

differential signaling差分信号

disabled interrupt禁用中断

distributed arbitration分布式仲裁

error control function错误控制功能

execute cycle执行周期

fetch cycle取周期

flit轻便

flow control function流量控制功能

instruction cycle指令周期

interrupt中断

interrupt handler中断处理器

interrupt service routine (ISR)中断服务程序（ISR）

lane车道

memory address register(MAR)内存地址寄存器

memory buffer register (MBR)内存缓存寄存器

multilane distribution多道分配

packets包

PCI Express (PCIe)

peripheral component interconnect(PCI)外围组件互连

phit

QuickPath Interconnect(QPI) QuickPath互连

root complex根联合体

system bus系统总线

思考题

3.1说出冯·诺依曼体系结构的三个关键概念。

3.2说明I / O地址寄存器和I / O缓冲寄存器之间的区别。

3.3禁用中断有什么缺点？ 请举例说明。

3.4计算机的互连结构（例如总线）支持什么类型的传输？

3.5列出并简要定义QPI协议层。

3.6列出并简要定义PCIe协议层。

习题

3.1 图3.4所示的假想机器同样有两个I/O指令：

0011 =从I/O装人AC

0111 =将AC内容存人I/O

在上述约定下， 使用12位的地址来标识一个特定的I/O设备。 给出下列程序的执行过程(用图3.5的格式）

(1)从设备5装人AC。

(2)与内存单元940的内容相加。

(3)将AC存人设备6。

假设从设备5提取的下一个价为3, 而内存单元940的内容为 2。

3.2 本书中使用6步来描述图3-5的程序执行， 讲解释这些步骤以说明MAR和MBR的作用。

3.3 另虑一个假想的32位微处理器采用32位的指令格式 ， 这种指令有两个部分 第1个字节包含操作码，其余部分是立即橾作数或橾作数的地址。

(a)最大可能扛搂寻址的存估器容社是多少（以字节为单位）？

(b) 讨论下面的饮处理器总线对系统的影响，

(1) 32位凡部地址总线和16位局部数据总线。

(2) 16位凡部地址总线和16位局部数据总线。

(c) 程序汁数器和指令寄存器需要多少位？

3.4 考虑一个假想的微处理器， 它产生16位地址（例如 ， 假设程序计数器和地址寄存器都是16位）， 井且有16位数据总线。

(a)如果处理器连接到"16位存储器＇， 那么它能直搂访问的战大存储器地址空间是多少，

(b)如果处理器连到., 位存储祥”、 那么它能立接访问的最大存储器地址空间是多少？

(c)结构上的什么特点允许处理器访问独立的 ·vo空间"?

(d)如果轮人和输出指令能够指定一个8位的VO端口号， 那么处理器能支持多少个8位的VO端口， 它能支持多少个16位的VO端口？诮Ill释。

3.5 芳虑一个32位饿处理器， 它有16位外部数据总线 ， 由SMH,的输人时钟驱动。 假设该处理器的总线周期的最小持续时间等于4个 输人时钟周期， 这个处理祥利）"总线能够维持的最大数据传输率是多少 （字节／秒），如果将其外部数据总线扩展为32位 ． 或使提供给处理器的外部时钟频率加倍， 能否提环它的性能，f/f陈述所做其他假设的理由， 并加以解释。 提示，确足每个总线周期所能传送的字节数。

3.6 ,; 虑一个计疗机系统 ， 它包括控创简中的键盘／电传打印机的VO模块。 下列寄存器包含在CPU中，

井且立接忐接到系统总线上。

INPR 输人寄存器(8位）

OUTR, 输出寄存器(8位）

FGI, 输人标志(I位）

FGO 输出标志(I位）

IEN 中断允许 (I 位）

从电传的按键输入到电传的打印机输出， 均由1/0校块控制。 电传能够将字母符号编钙成8位的字 ， 并将8位的字解砃为字母符号。

l•I描述该处理器如何能通过使）II本题中给出的前4个寄存器来完成与电传的输人 输出。

(bl描述这一功能如何能通过使用JEN 更打效地完成。

3.7 芍恁两个微处理器， 除了一个使用8位宽的外部数据总线而另一个使用16位宽的外部数据总线之外两行没有其他不向 它们的总线周期也是等长的。

(,)假设所有指令和数据都处两字节长， 则它们的朵大数据传输率差几倍？

bl假设指令是一字节长， 数据是2字节长， 觉复上述问题。

3.8 l!l 3-26表示一种分布式的仲栽方法， 它可用于 M"t,;b"'I的栋旧总线方式中。 各中元按优先级次序在物理上以菊花链的方式迕接。 图上最左边的小元持续接收总线优先权输人(BPRN)信号， 表示没打更环优先级的单元需要使用总线。 如果这个小元不需荽使用总线， 则它卢明自己的总线优先级输出JBPRO)线。 在时钟周期的开始 任何单元都可以使其BPRO信另线为低来访求总线的控利。 这使菊花链中的 下一单元的 BPRN 线为低， 而这个单元也相应地降低自己的BPRO线。 因此 信号就沿汗／菊花链传播。在链式反应的末尾，应只有一个车元， 它的BPR,, 才，效， 而

BPRO无效， 于是这个爪元获得了优先权。 如果在总线周期的开始， 总线处于空闲状态(BUSY线无

效）， 则获得优先权的牛元可以通过声明BUSY线来夺取总线的控利权。

BPR信号从儿有社环优先级的单元传播到儿有最低优先级的年元飞耍一定的时间。 这段时间必须比时钟周期小吗，请解释原因．

3.9 VAX SB\总线使用分布式同步仲纹方案。 每个SBI设备（例如处理器， 存储器、 VO校块）有唯一的优先级， 而且被分配一根独立的传输请求(TR)线。 SB\打16根这样的线(TRO、 TRI 、 , TR\5), 其中TRO的优先级最优． 当一个设备想要使Ill总线时， 它通过在当前时间片中卢明TR线来预约未来的时间片。 在当前时间片结束时， 每个具打预约请求的设备检查TR线 ， 其中具打最高优先级的设备使用下一个时间片。

最多可打17个设备迁搂到总线上， 优先级为16的设备没有TR线， 请说明原因。

第四章

计算机的存储器虽然从概念上来看比较简单， 但是从计算机系统的类型、 技术 、 组织、 性能和价格几方面的特点来看，存储器的范围或许是最广的。目前没有一种最佳的能满足计算机系统对存储器需求的技术。所以， 计算机系统通常配备分层结构的存储子系统 ， 一些在系统内部 （山处理待直搂存取）． 一些在系统外部（处理器通过VO校块存取）。

本章和下一产将重点论述系统内部存估器部件， 而第6未将七门论述外部存储器。 本在甘先介绍计n:机存忧器的关键特性． 剩余部分讨论所打当代计j'J:机系统所必备的部件--,矶J,e存 估器。

4.1 计算机存储系统概述

存储系统的特性如果我们按照关键特性对存储系统进行分类， 那么计算机存储器的复杂问题就会变得更易于管理，表4-1列出了存储系最重要的一些特性。

存储位置是指存储器处于计算机的内部或外部。内部存储器通常指主存，但还有其他形式。处理器需要有自己的局部存储器，它们以寄存器的形式存在（如图2.3所示）。进一步，我们就会明白处理器的控制器部分也需要有自己的内部存悄器。 我们将在后面欢节对这两类存忧骈进行分别讨论。 coch, 是内存存储器的另一种形式。 外部存储器巾外围存估设备（如磁盘 ， 磁带等）组成， 处理器可以通过1/0控制器访问它们。

存估器的一个明显特性是存储容量(copoe,ty) 。 对千内部存恬器， 存储容员通常用字节(I 字节:8位）或字来表示， 许通的字长为 8位 、 16位或32位。 外部存储器的存估容拭通常也用 字节来表示。

一个与之相关的概念是传输单元(,.,,. of t<oosfm) 。 对于内部存忧器 ， 传输单元等同于输人和输出到存储器模块的数据线数， 它守千字长， 但通常更大． 女1164位 、 128位或256位。 为了说 明这一点 ， 我们引人三个与内部存储器相关的概念：

• 字存储器组织的 “ 自然 单元。 字长通常与一个整数的数据位数和指令长度相等 但也有很多例外。 例如 ， CRAY C90 (一种较老的栈型CRAY超级计符机）有64位的字长， 但它用 46位表示整数。 而'"'"',86体系结构有各种指令长度． 用多个字节表示， 但其机器的字长为32位。

• 可寻址单元：在某些系统中 ． 可寻址单元是字 ， 但许多系统允许在字节级上寻址。 在任何情况下， 地址位长度A和可寻址的单元数N之间的关系为，2':N。

• 传输单元 对于主存估器 ， 这是指每次读出或写入存储器的位数。 传输单元不必等千一个字或一个可寻址单元。 对千外部存储器， 数据的传送经常是以比一个字大得多的单元 来传送， 这就是所谓的块。

不同种类的存储器之间的另一个区别是数据单元的存取方法(meth心ofocce豁iog)不同 ， 存 取方法包括如下四类：

顺序存取：存估器组织成许多称为记朵的数据单元． 它们以特定的线性序列方式存取。 存储的地址信息用于分隔记录和帮助索引。 采用共亨读－写结构． 经过一个个的中间记录，从当前的存储位置移动到所要求的位壮， 因此 ， 存取不同记录的时间相差很大。 第6 章中讨论的磁带机采用的是顺序存取方式。

．直接存取：同顺序存取一样 ， 直搂存取也采用了共亨读－写结构。 但是， 单个块或记录有基于物理存储位置的唯一地址。 通过采用直接存取到达所需的块处， 然后在块中顺序搜索 计数或等持， 最终到达所要求的位置。 同样 ， 存取不同记录的时间相差很大。 第6章中讨论的磁盘机系统采用的是直接存取方式。

• 随机存取：存储器中每一个可寻址的存储位置有唯一的物理编排的寻址机制。 存取给定存储位置的时间是固定的． 不依赖干前面存取的序列。 因此，任何存储位置可以随机选 取 直接寻址和存取。 主存和某些高速缓存系统采用随机存取方式。

• 关联存取：这是一个随机存取类的存储器， 它允许对一个字中的某些指定位进行检查比较 行是否与特定的样式相匹配， 而且能同时在所存字中进行。 因此， 字是通过它的内 ，

容而不是它的地址进行检索。 与普通的随机存取存取器相同， 每个存估位置有自己的寻 址机制， 并且检索时间是固定的， 不依赖千存储位灶或前面的存取方式。 点速缓存可以采用关联存取。

从用户的观点来石， 存储器两种ltl须要的特性是容员和性能（严”叩nooce) , 通常需要芳虑 3种性能参数：

• 存取时间（延迟｝ 对干随机存取存储器， 这是执行一次读或写操作的时间， 即从地址传送给存估器的时刻到数据已经被存恬或使用为止所花的时间。 而对于非随机存取存估祥， 存取盯间是把读－写结构定位到所需要的存估位登所花费的时间。

• 存储周期时间．这个概念主要用干阅机存取存储器， 它是存取时间加上下一次存取开始之前所需要的附加时间。 这里附加时间用于瞬变的信轩消失或数据破坏性读后的再生。俙要注意， 存储周期时间姑与系统总线打关， 而不是与处理器相关。

传输率 这是数据传人或传出存估单元的速率。 对于随机存取存储器， 它等于 "\/周期 时间”。 而对于非随机存取存估器 ， 有下列关系



其中 Tn＝读或写N位的平均时间

Tn＝平均存取时间

"=位数

R=传输率， 单位为bl, (位／秒）

存储器有许多种物理类型． 目前垃常用的打半导体存储器． 用于磁盘和磁带的磁表面存估器以及光学和磁－光存储器。

数据存储的儿个物理特性很瓜要。 在易失性存储器中 ｀ 当电游开关断开时， 信息自动衮减或丢失。 而在非易失性存储器中． 信息一且记录， 就会保招到下一次打怠改变它时为止． 不需要电说来维持信息。 磁表面存忧器是非易失性的。 半导体存忧器可以是易失性的． 也可以是非易失性的。 不可擦除存储器不能修改. 除非破坏存估单元， 这种类型的半导体存储器被称为只读存估器 (ROM)。 当然， 不可擦除存储器也必定是非易失性的。

对于闭机存取存储楛 ． 存钻单元的组织是一个关键的设计问题。 本文中组织的意思指通过物理排列位来形成字。 如第5东介绍的那祥． 并不总是使用简单的排列。

存储器层次结构计环机中存储器的设计限制可以归纳为3个问题．容让有多大？速度有多快？价格有多货？容从大小似乎并没有限制， 不忤容从多大， 总要开发应用程序去使用它。 速度多快的问题从 某种意义上来说更容易回答。 为了获得坎佳性能， 存储器的速度必须能够跟上处理器的速度。 也 就是说， 当处理器在执行指令时， 我们并不期望它因为等待指令或操作数而暂停执行。 最后一个 问题也必须考虑。 对于实用系统， 存估器的价格相对于其他组件来说必须是合理的。

正如我们所预料的， 在存储器的3个关键特性容扯、 存取时间和价格之间需要进行权衡。 用来实现存估系统的技术打多种， 在这一系列的技术中都存在如下关系：

• 存取时间越短， 平均每位的花费就越大。

• 存储容让越大， 平均每位的花费就越小。

• 存储容杖越大， 存取时间就越长。 设计者面临进必两难的局面是明显

设计者面临进退两难的局面是明显的。设计者想要使用存储器技术提供大容从的存储器， 因为这既涡足容fi!的要求， 也使每位的价格低。 然而， 为了涡足性能的要求， 设计者又不得不使用昂贵的、 相 对来说容狱较小而存取速度较快的存储待。

解决这个难题的办法不是只依赖单一的存储部件或技术， 而是采用存储器层次 结构(mem叩朊,a叫hy) 。 图4-1给出了一种典型的尼次结构， 随fj臣次的下降， 我 们会发现：

(a)每位价格下降，

(b)容狱均大；

(e)存取时间变长，

(d)处理器访问存储特的频率降低。

因此，容量较小、价格较贵、速度较快的存储器可作为容量较大、价格较便宜、速度较慢的存储器的补充。 这种组织方法要取得成功的关键是最后一项(d). 即降低访问频率。 我们将在本齐后面部分讨论coche和第8l;t讨论虚拟存估器时详细介绍这个概念， 这里只做简单解释。

从原理上讲，使用二级存储器可以减少平均存取时间， 但此时要求条件(,)到条件(d)都满足。 T,• 乃通过采用各种技术， 出现了一系列满足条件(a)到条件(c)的存储系统， 幸运的是， 条件(d)通常也能满足。

条件（d）有效的基础是访问的局部性原理[DENN68]。在程序执行的过程中，处理器倾向于成簇（块）地访问存忧器中的指令和数据。 程序通常包含许多迭代循环和子程序。 一旦进人一个循环或子程序， 则会项及访问一小组指令。 同样 ． 对千表和数组的操作包含存取一簇簇的数据。 在一段较长的时间中 ，使用的簇是变动的， 但在一小段时间内 ， 处理器主要访问存储器中的固定簇。

例4.1 假设处理器支持二级存忧器结构。 第一级存估器包含1000字， 存取时间为0.01µ,, 第二级存储器包含100 000字， 存取时间为0.1 卢。 假定要访问的字在第一级存估器中 ， 则处理器能直桵对它进行存取。 如果字在第二级存储器中 ， 那么该字将首先被传送到第一级， 然后处理帮再对它进行存取。 为了使问题简化， 我们忽略处理器用来判断要访问的字在哪一级所需要的时间。 图4-2给出包含此问题的曲线的基本形状， 该图显示了二级存储器结构下平均存取时间和命中率/I之间的函数关系， 其中H被定义为在较快存忧器（如cache)中完成的存取占所有存储器存取的百分比， r, 是访问第一级存估器所需要的时间， 而r, 为访问第二级存恬器所需要的时间气可以行,1,. 在第一级存忧器中的访问百分比越环， 总的平均访问时间就越接近访问第一级存储器所需要的时间， 而不是第二级的时间。

在此例中 ， 假设95%的存估器访问都可以在cache中找到 ， 那么平均访问一个字的时间可以表示为：

（0.95）（0.01us）+（0.05）（0.01us+0.1us）= 0.0095+0.0055 = 0.015us

正如我们所期望的， 平均访问时间更接近千0,01us , 而不是0.1us。

因此，通过层次结构组织数据，有可能使访问较低层存储器的百分比低于访问其上层存储器的百分比。考虑前面已给出的二级存储器的例子，让第二级存储器包含所有程序指令和数据。当前簇可以临时调入第一级存储器。有时，第一级中的某个簇将不得不被交换回第二级存储器，以便为所需要的新簇腾出空间。然而，平均来说，处理器大部分是对第一级中的指令和数据进行访问。

这一原理可以应用于多级存储系统。如图4-1所示的层次结构所建议的一样，速度最快、容量最小而价格lil科的存储器是处理器内的寄存器。 通岱， 一个处理器包含几十个这样的寄存 器， 但也打一些机器包含数百个寄存器。 主存储器是计算机中主要的内存系统， 主存中的每个存储位坟都打唯一的地址。 主存通常用更快速度 、 更小容员的each, 对其进行扩充。Cache对于程序员乃至处理器来说是透明的， 它在主存和寄存器之间分段传送数据以提高性能。

上面所介绍的三种形式的存储器通常都是易失性存储器， 普忐采用半导体技术。 这3层存钻 器的使用导致了各种类型的半导体存储器的开发， 它们的速度和价格各不相同。 数据仵各种外部的 、 大容址的存钻设备上能存储得更持久． 最常用的是硬盘和可移动媒体， 如可移动磁盘 磁带和光存忧器。 外部非易失性存储器也称为辐助存储器或输存”釭'"d,,y皿m叩，或,.,,.i;,,y memo,y), 它们常用于存储程序和数据文件， 以文件或记录的形式而不是以一个一个的字节或字的形式为程序员所使用。 磁盘也可以用于主存忧器的扩展， 称为虚拟存忧器． 这将在第8市中讨论。

层次结构中还包含其他形式的存储器。 例如 ， IBM大型机通常包含了一种称作为扩展存钻器 的内存储器， 它采用一种比主存储器要慢且更便宜的半导体技术。 严格来讲， 这种存储器不列人层次结构中， 而只是一个分支 数据可以在主存和扩展存忧器之间进行传送， 但不能在扩展存估符和外存恬器之间进行传送。 其他形式的辅存包括光盘和 “磁－光盘” 设备。 最后 ． 可以仆层次 结构中以软件形式有效地加入附加层。 主存储器中的一部分可用做缓冲区｀ 暂时保存写人磁盘中的数据． 这种技术有时称为磁盘高速缓存。， 它用两种方法改进性能：

• 磁盘写人是以簇的形式进行的。 一次传输的数据肚较大， 而不是很多次小的数据传送， 这改善了磁盘的性能｀ 减少了对处理器的占用。

• 某些指定输出的数据在转存到磁盘之曲可被程序访问。 这样， 可以快速地从软件I，.速缓 存中检索． 而不是从相对较慢的磁盘中检索。

附录4A论述了多级存储器结构对性能的影响。

4.2 cache存储器原理

高速缓冲存储器旨在使存储器的速度逼近可用的最快存储器的速度，同时以较便宜的半导体存储器的价格提供一个大的存储器容量。图4.3a说明了这一概念，图中有一个相对大而慢的主存。加之一个小而快的cache。cache中存放了主存储器的部分副本。当CPU试图访问主存中的某个字时，首先检查这个字是否在cache中，如果是，则把这个字传送给CPU；如果不是，则将主存中包含这个字固定大小的块读入cache中，然后再传送该字给CPU。因为访问的局部性，当把某一块数据存入cache，以满足某次存储器的访问时，CPU将来还很有可能访问同一存储位置或该数据块中的其他字。

图4.3b描述了多级cache的使用，其中，第二级cache比第一级cache要慢，但通常存储容量较大；而第三级cache比第二级cache慢，但通常存储容量要大。

图4.4描述了cache/主存系统的结构。主存估器由多达,. 个可寻址的字组成， 每一个字打唯一的n位地址。 为了实现映射， 我们将主存估器行成是由许多定长的块组成， 每块有 K个字。 即有M=2"/K个块。 而«ch, 包含m个块， 称作行。， 每行包括k个字和儿位标记。 每行还包括控制位（图中没有给出） ， 如用作判断装入each,中的行是否被修改的控制位。 行的长度． 不包 含标记和控制位， 称为行大小(H"'亚,) 。 行大小可以小到32位， 其每个 ”字 就是单个字节． 此时． 行大小是4个字节。 行的数杜远远小于主存储器块的数目(m«M) 。 任何时候， 只打主 存储器块的子集驻留在,oche行中。 如果要读取主存估器块中的某个字， 则包含该字的块将被传送到c,che的一个行中。 由于块数多于行数 ． 所以单个行不可能永久地被某块专用。 因此， 每行 都有一个标记(1屯） ． 用来识别当前存储的是哪一块。 这个标记通常是主存恬器地址的一部分． 这将在本节后面加以讨论。

图4-5说明了一个读操作。 处理器产生一个要读取字的地址RA, 如果这个字在c,che中， 则把它直接传送给处理器。 否则， 将包含这个字的块装人c,che中， 然后再传送给CPU。 图4-5表 不， 最后两步操作是并行进行的， 这在图4.6所示的当代cache经典组织中有所反映。 在这种组织结构中， C虹加经数据线， 控制线和地址线连接到处理器， 数据线和地址线也分别与数据缓冲器和地址缓冲器相连， 这些缓冲器都接到系统总线上， 从而与主存 连接。 当c,chc命中时， 数据和地址缓冲器都不启用， 通信只在处理器和cac加之间进行， 此时系统总线上没有信冉传输。 当 mhe未命中时， 所需求的地址被加载到系统总线上， 数据通过数据缓冲器提交给c,d,e和 CPU。 在其他的组织结构中， 通过所打数据线、 地址线和控制线， c,ch, 直接介于处理器和主 存之间 ， 在这种情况下 ， 当mhe未命中时 ， 要读取的字先被装人mh,, 然后再由cache传送给CPU。

有关使用c,ch,的性能参数讨论可参见附录4A。

4.3 cache 的设计要素

本节给出了cache设计参数的概述井报告了某些典型结论。 我们偶尔会谈及cache在高性能计算（high-performance,HPC）中的使用。HPC涉及超级计算机（supercomputer）和超级计打机软件， 主要应用于科学计11. 因为这类应用中包含了大拭的数据、 向坎和矩阵 的计11:. 以及井行n法的使用。 为HPC设计的 c>che与为其他硬件平台和应川所设计的c缸h, 是非常不同的。 的确 ， 许多研究人员已经发现HPC应Ill在使用了cache的计疗机体系结构上性能会变控[ BAIL93] 。 而另一些研究人员也 已指出 ， 如果应用软件适合使用c叱h,, 那么切ch, 层次将对改善性能很有帮助[WANG99, PHESOI]气

尽管已有大量的cache实现方案 ， 但只有几个基本要素用于区别和分类cache的体系结构，表4.2列出了关键的要素。

cache地址几乎所打的非嵌人式处理器以及很多嵌人式处理器都支持虚拟内存。 虚拟内存的概念将在 第81,\'中讨论。 本版上讲， 湿拟内存是一种内存扩充技术， 这种技术不会使主存物理地址空间大 小发生改变 ， 但允许程序在逻机上访问更多的地址。 当使用虚拟内存时， 机器指令的地址域包含 虚拟地址。 为了从主存中进行读写操作， 硬件存储器管理小元(MMU)将每个虚拟地址翻译成主存中的物理地址．

当使用虚拟地址时，系统设计人员可能选择将cache置于处理器和MMU之间，或者置于MMU和主存之间。如图4.7所示。逻辑cache，也称为虚拟cache，使用虚拟地址存储数据。处理器可以直接访问逻辑cache，而不需要通过MMU。而物理cache使用主存的物理地址来存储数据。

逻辑cache的一个明显优势是其访问速度比物理cache快，因为该coch, 能够在 MMU执行地址翻译之前作出反应。 而其不足之处在于大多数虚拟存储系统为每一个应用程序提供相同的戍拟内存空间。 也就是说， 每个应用程斤都可以从地址为0的虚拟内存片始。 因此． 相同的虚拟地址在两个不同的应用程序中涉及不同的物理地址。所以， c,chc存储器必须用每一个应用程序的上下文开关对其进行完全刷新． 或者为co,h, 的每一行珔加额外的几位来标记与该地址相关的虚拟地址。

逻钳'""'和物理coch, 的比较是一个很复杂的问题， 在本书中不进行过多的讨论。 想了斛更多． 可参芳[CEKJ.97]和[JAC008]。

cache容量我们前面已经对coch, 容从进行过讨论， 表4-2中也有提到。 我们希望each, 的容扯足够小．以至于整个存储系统的平均每位的价格接近于单个主存储器的价格． 同时我们也希望each, 足够 大． 从而使得骼个存储系统的平均存取时间接近于单个coch, 的存取时间。 还有儿个减小,och, 容从的动机。 也,h, 越大， 寻址所需要的电路门就会越多． 结果是大 的mh, 比小的稍慢． 即使 是采用相同的集成电路技术制造井放在芯片和电路板的同一位登。 coch, 容拭也受芯片和电路板面积的限制， 因为cech, 的性能对工作负载的性能十分敏感， 所以不可能有 “最优” 的coch, 容 ft。 表4-3列出了过去和当前的某些处理器的coch,容队。

映射功能由于cache的行比主存储器的块要少，因此需要一种算法来实现主存块、cache行的映射。而且，还需要一种方法，确定当哪一块占用了cache行。映射方法的选择决定了cache的组织结构，通常采用三种映射方法：直接映射、全相连映射和组相连映射。下面我们依次讨论这三种方法，分析每种方法的通用结构及具体的例子。

例4.2 对于所有这三种映射方法，该例子中都包含下列条件：

Cache能存储64KB

数据在主存和cache之间以每块4字节大小传输。这意味着cache被组织成16K=2^14行，每行4字节

主存容量为16MB，每个字节直接由24位的地址（2^24=16M）寻址。因此，为了实现映射，我们把主存看成是由4M个块组成，每块4字节。

直接映射直接映射是最简单的映射技术，将主存中的每个块映射到一个固定可用的cache行中。直接映射可表示为：



其中：i=cache行号

j=主存储器的块号

m=cache的行号

图4.8a给出了主存中前m块的映射情况。如图所示，主存中的每一块映射到cache中的唯一行，然后接下来的m块依次映射到cache中的相应位置。也就是说，主存中的Bm块映射到cache中对应的L0行，Bm+1块映射到L1行，等等。

映射功能通过主存地址很容易实现。图4.9描述了基本的映射机制。为了访问cache，每一个主存地址可以看成是由三个域组成。最低的w位标识某个块中唯一的一个字或字节；在当代大多数机器中，地址是字节级的。剩余的s位指定了主存2^s个块中的一个。Cache逻辑将这s位转换为s-r位（最高位部分）的标记域和一个r位的行域，后者标识了m=2^r个cache行中的一个。小结如下：

地址长度=（s+w）位

可寻址的单元数=2^（s+w）个字或字节

块大小=行大小=2^w个字或字节

主存的块数=2^（s+w）/2^w=2^s

Cache的行数=m=2^r

cache的容量=2^（r+w）个字或字节

标记长度=（s-r）位

例4.2a 图4.10表示了一个使用直接映射的实例系统。在这个例子中，m=16K=2^14，

而且i=j mod 2^14，映射变为：

|  |  |
| --- | --- |
| Cache行 | 主存块的起始地址 |
| 0 | 000000,010000，...，FF0000 |
| 1 | 000004,010004，...，FF0004 |
| ... | ... |
| 2^14-1 | 00FFFC，01FFFC，...，FFFFFC |

注意， 映射到相同行号的两块不会有相同的标记数。 因此， 开始地址为000000,010000, FFOOOO的块对应的标记数分别为00,0(,···, FF 。

回头来再和甘4-5,读操作的流程是这样的, coch, 系统用 24位地址表示， (4位的行号用来做特定行的索引。 如果8位标记数与当前存储在该行的标记数相匹配， 则用2位字号来选取行中 的4个字节。 否则 ， 22位的标记加行号域被用来从主存中取出一块。 取主存块所用的实际地址 姑22位的标记加行号再接两位0, 因此， 在块的边界起始处读取4个字节。

这种映射的结构是把主存中的块分配给如下所示的cache行中。

|  |  |
| --- | --- |
| Cache行 | 被分配的主存块 |
| 0 0, *m*, 2*m*, c, 2*s* - *m* | 0 0, *m*, 2*m*, c, 2*s* - *m* |
| 1 1, *m* + 1, 2*m* + 1, c, 2*s* - *m* + 1 | 1 1, *m* + 1, 2*m* + 1, c, 2*s* - *m* + 1 |
| ... | ... |
| *m* - 1 *m* - 1, 2*m* - 1, 3*m* - 1, c, 2*s* - 1 | *m* - 1 *m* - 1, 2*m* - 1, 3*m* - 1, c, 2*s* - 1 |

因此 ， 采用部分地址作为行号提供了主存中的每一块到c,che的唯一映射。 当一块读人到分 配给它的行时， 必须给数据做标记， 从而将它与其他能装人这一行的块区别开来。 最奋的'-, 位用来做标记。

直接映射技术简单．实现起来花费也少。其主要缺点是：对于任意给定的块．它所对应的 e,ch, 位置是固定的。因此，如果一个程序恰巧煎复访问两个需要映射到同一行中且来自不同块 的字亨则这两个块将不断地被交换到coche中.co,h, 的命中率将会降低（一种所谓的抖动现象）。

一种降低缺失开销的办法是保存被丢弃的数据以备再次需要用到它。 因为被丢弁的数据已经被取进mh, 中过， 因此再次使用的开销比较小。 使用V,et,mmh, 可以实现这种资源顶复利用机利。 妏初提：Ii V忙t,mmh, 这一概念是为了减少扛接映射each, 中冲突缺失的次数， 井且不影响其快速存取的时间。 V灿meoch, 是一种全相联"""· 其存储容狱一般为4 -16个each, 行, l','于使用 I述择性的v;ct而高速缕存戌拟器 且接映射的第一级each, 和下一级存恬器之间。 这一概念将在附录F中进行探究。

例4.2b 图4.12给出了使用全相联映射的例子。主存地址由22位标记和2位的字节号组成。22位标记必须与32位数据块一起存储在cache行中。注意，地址的最左(lil高） 22位形成标记。 因此 ， 24位十六进制地址16339C有22位标记OS8CE7 。 这小二进制表示法很容易行出

存储地址 0001 0110 0011 0011 1001 1100（二进制）

1 6 3 3 9 C

标记（最左22位） 00 0101 1000 1100 1110 0111（二进制）

0 5 8 C E 7

注意：地址中无对应行号的字段，所以cache中的行号不由地址格式决定。 总结如下：

• 地址长度= (s+w)位

• 可寻址的单元数=2^(s+w)个字或字节

• 块大小＝行大小＝2^w个字或字节

• 主存的块数=(2^(s+w))/2^w=2^s

• c,che的行数＝不巾地址格式决定

• 标记长度=s位

对于全相联映射，当新的块读人c,che中时，替换旧的一块很及活。忤换符法本节后面将要讨论，它用来使命中率朵大。全 相联映射的主要缺点姑份要复杂的电路来井行检在所有的cache 行标记。

组相连映射 组相联映射是一种折中方法，它既体现了直接映射和全相联映射的优点，又避免了两者的 缺点。

在组相联映射中,each, 分为u个组，每组包含k个行，它们的关系为



其中：i=cache组号

j=主存块句

m=cache的行数

v=组数

k=每组中的行数

这被称为k路组相联映射。采用组相联映射，块B,能够映射到组）的任意行中。图4-13,给出了主存中前u块与cache行的映射关系。在全相联映射中，每一个字映射到多个cache行中。而对于组相联映射， 每一个字映射到特定一组的所有c,che行中， 于是， 主存中的B, 块映射到 第0组， 如此等等。 因此 ， 组相联映射cache在物理上是使用了u个全相联映射的mhe。 同时．它也可行作为K个立接映射的c,che的同时使用， 如图4-136所示。 每一个立搂映射的mhe称为 路 包括“个cache行。 主存中首u个块分别映射到每路的o行中， 接下来的，个块也是以同样的方式映射， 后面也如此。 五接映射一般应用于轻度关联(k伉较小）的情况， 而全相联映射应用 于环度关联的情况[JAC008) 。

在组相联映射中， C邸h, 控利逻铅将存储地址表示为三个字段：标记、 组和字。 长度为d位的组字段指定了,=2"个组中的唯一一个组， 标记字段和组字段共长s位， 用以标明主存中2'个块中具体某一块。 图4-14描述了each, 控制逻辑。 在全相联映射中， 主存地址中的标记字段很长 而且还必须与c邸加中每一行匹配。 而在K路组相联映射中， 主存地址中的标记字段要短很多 而且只需与某一组中的K行匹配。 总结如下：

• 地址长度=(s+w)位

• 可寻址的单元数=2"一个字或字节

• 块大小＝行大小=2一 个字或字节

• 主存的块数=2""/2"=2'

• cache中每组的行数=k

• 组数=v=2"

• cache中的行数=m=kv=k\*2^d

• cache存储容量=k\*2^d字或字节

• 标记长度=（s-d）位

例42c 图4-15给出了一个用组相联映射的例子， 其中每一组有两行， 也就是二路组相联。 13位长的组号标识了each, 中唯一的两行组， 也给Iii了用2"取校后的主存块号。 这确定了块到行的映射。 因此 ， 主存中的块000000, 00000 , ·.. , FFSOOO映射到,ache中的第0 组， 其中每一块都能装人该组两行中的任怠一行。 注戏， 两个映射到同一cache组的块不可能具布相同的标记数 对于读操作， 用13位组号检查确定组地址， 组中的两行与被存取地址的标记数进行匹配检查。

在v=m, k=l的极蠕情况下， 组相联技术简化为立接映射。 而对于,= I 、 k:m的情况． 七又会等同于全相联映射。 每组两行(,:m/2, k:2)是最岱用的组相联结构。 与直接映射相比 ， 它明品地提环了命中率。 四路组相联(,:m/4, k:4)用相对较少的附加成本使命中率有一些提l1, [MAYB84, HILL89] 。 继续泭强每组的行数对cache命中率的提奋几乎没什么效果。

图4-16给出了组相联c邸h,性能的一个榄拟研究结果， 图中显示了不同切,h, 容拭对cad比 性能的影响[GENU04]。 我们注意到， 当cache的容肚达到64kB之前， 直接映射和二路组相联 映射的性能区别是非常显若的。 同时也注意到， 二路组相联和四路组相联在cache容从均为4kB 时的性能差别要远小千cache容员从4kB变到SkB时的经别。 由于cache的复杂性与相联性成正比例，因此，在这种情况下，将cache容量增大到8kB甚至16kB都是没有道理的。最后需要注意的是，当cache容量超过32kB时，cache容量的增加对提高性能的作用并不明显。

图4.16是基于对GCC编译器执行的模拟。不同的应用可能会产生不同的结果。例如，【CANT01】报告了使用很多CPU 2000 SPEC基准程序时的cache性能结果。【CANT01】中比较命中率与cache容量的结果与图4.16中的基本相同，但是指定的值有些不同。

4.3.4 替换算法

一旦cache行被占用． 当新的数据块装人cache中时 ， 原存在的块必须被替换掉。 对于直接映射， 任意特殊块都只有唯一的一行可以使用， 没打选择的可能。 对于全相联映射技术和组相联映射技术， 则需要一种忤换＂：法。 为了获得高速度 ， 这种钉法必须由硬件来实现。 人们尝试过许多算法， 下面介绍最贷用的4 种打法。 可能坎有效的算法姑杖近最少使用的算法(LRU) 井换掉那些在cache中最长时间未被访问过的块。 对于两路组相联． 这种方法很容易实现， 每行包含一个USE位。 当某行被引用时， 其USE位被灶为'· 而这一组中另一行的USE位被投为0。 当把一块读人到这一组中时 ， 就会替换掉USE位为0的行。 由于我们假定越是朵近使用的存估单元越打可能将被访问， 因此， LRU会给出最佳的命中率。 对于全相联mhe, LRU也相对容易实现 环速缓存机制会为mhe中的每行保留一个单独的索引表。 当某一行被访问时， 它就会移动到农头 ， 而在表尾的行将被竹换I伈因为其实现简单， LRU是目前使用最广泛的替换算法。

另一种可能的算法是先进先出（FIFO）：替换掉那些在cache中停留时间最长的块。FIFO采用时间片轮转法或环形缓冲技术很容易实现。还有另一个可能算法是最不经常使用（LFU）：替换掉cache中被访问次数最少的块。LFU可以用与每行相关的计数器来实现。第四种算法是一种不基于使用情况的技术（不是LRU,LFU,FIFO,或其他变体），它是在候选行中任意选取，然后进行替换。模拟实验结果表明，随机替换算法在性能上只稍逊于基于使用情况的算法【SMIT82】。

4.3.5 写策略

当驻留在c眈加中的某块要被杆换时， 必须巧虑两点。 如果cache中的原块没打被修改过，那么它可以被直接替换掉， 而不需要邓先写回主存。 如果在也,he某行中至少在一个字上进行过 写操作， 那么在杆换掉该块之前必须将该行写回主存对应块， 以进行主存更新。 各种可行的写策 略都对性能和价格进行了权衡， 但还存在 两个争论的问题。 首先， 有一个以上的设备已经访问了 主存储器。 例如， VO校块可能且接读／写存估器。 如果一个字只在cache中修改过， 那么相应的 存恬器字就是无效的。 进一步． 如果某VO设备修改了主存储器 ， 则c,che中的字是无效的。 当 多个处理器连接到同一总线上 ， 井且每个处理器都有自己局部的"""时， 则出现了更复杂的问题。因此，如果一个cache中修改了一个字，那么可以设想在其他cache中该字是无效的。

最简单的技术称为写直达(wrile ihroogh) 。 采用这种技术， 所有写操作都同时对主存和 '"加进行， 以保证主存中的数据总是有效的。 任何其他处理器－莉速缓存模块监视对主存的访问 都姑维护它自己coche的一致性。 这一技术的主耍缺点怂产生了大拭的存储通信扯 ， 可能引起瓶颈问题。 另一种技术称为写回法(wrileb缸k), 它减少了主存的写入。 使用写回技术时， 只更新"""中的数据。 当更新操作发生时， 需要设性与该行相关的脏位(dirty bi1)或使用位 (, 妃加） 。 然后 ， 当一个块被替换作时 当且仅当脏位被灶位时才将它写回主存。 写回的缺点是 部分主存数据是无效的， 因此f/0校块的存取只允许通过c叱he进行， 这就使得电路设计更加复杂而且存在潜在的瓶颈问题。 经验表明 ， 写操作占存储器操作的15% [SMIT82] 。 然而， 对 于HPC应用， 这个值可接近33% (向扯－向凡乘法） ， 甚至可店达50% (矩阵转'{1.) 。

例4.3 考虑一个行大小为32字节的,,che和一个传送一个4字节字用时30,s的主存。 co,he的任意行被忤换之拊至少巳被写过一次， 如果要使写回法比写且达法更扂效． 在被拧换之前平均每行被写的次数是多少？

采用写回法时，每一个脏行只在交换时写回主存一次， 需要8x30:240"'。 而采用写且达法时，每一次更新mhe中的某行都荽求布一个字写到主存， 耗时30as。 因此， 如果行换出之前写入平均超过8次的话， 则写同法更布效。

在不止一个设备（通常是处理器）打coche且共享主存的总线结构中出现了一个新的问题。 如果某个cache中的数据被修改， 则它不但会使主存中的相应字无效， 而且也会使其他mhe中的对应字无效（如果其他coche中恰巧也打这个字） 。 即使采用 “ 写直达” 策略， 其他c配he也 可能包含无效的数据。 防止这个问题的系统被说成是维护c眈he的一致性。 保证cache一致性的方法有：

• 写直达的总线监测：每个c,ch, 控制器监视地址线 ， 以检测总线的其他主控者对主存的写操作。 如果有另一个总线主控者向共享存储单元写人数据． 而这个单元内容同时驻留在cache中． 则该c,ch, 控制器使c叫h, 中的这一项无效。 这一策略要求所有c吐e控制器都使用写直达策略。

• 硬件透明 使用附加的硬件来保证所打通过mhe对主存的修改反映到所有coche中。 因此 如果某个处理器修改了自己coche中的一个字， 则同时会修改主存对应单元， 任何其 ，

他妞che中相同的字也同时会被修改。

• 非cache存储器：只有一部分主存为多个处理器共享 ， 这称为非coche。 在这样的系统 中 所有对共享存储器的访问都导致coche缺失， 因为共享存储器中的数据不会复制到cache中。非cache存储器能采用片选逻辑或高地址位来标识。

coche一致性是一个活跃的研究领域， 我们将在第五部分对其进行深人探讨。

行大小

另一个设计要素是行大小。当一个数据被检索并放入cache中时，所需的字和一些相邻的字都会被取出。当数据块由很小变得较大时，命中率刚开始会因为局部性原理而增加。局部性原理是指：被访问字附近的数据很可能会在不久的将来被访问到。随着块大小的增加，更多有用数据被装入cache。但是，当块变得相当大，并且使用新取信息的概率变得小于重用已被替换掉的伯息概率时，命中率开始下降。块的两个特殊作用如下：

• 较大的块减少了装人coche中的块数。 因为每个新块都会覆盖掉原来coche块中的内容， 少t..t的块导致了装人的数据很快被改写。

• 当块变大时， 每个附加字就会离所需字更远， 因此被使用的可能性也就更小。

块大小与命中率之间关系复杂， 它取决于特定程序的局部性特征， 目前还没有找到确定的最优值。 块大小为8 -64B时 ， 比较接近最优俏[ SMIT87, PRZY88, PRZY90, HAND98] 。 对于 HPC系统来说， 最常用的行大小是64B和128B。

cache数目砓初引人cache时， 系统通常只有一个coche。 近年来， 使用多个c眈he已经变得相当许凸。 我们所考虑设计问胜的两个方面是, each, 的级数以及采用统一或分立的cache。

多级cache由千织成度的提环， 将c邸如与处理器灶于同一芯片（片内coche)成为可能。 与通过外部总线连接的coche相比 ， 片内coche减少了处理器在外部总线上的活动， 从而减少了执行时间， 全面提向了系统性能。 当所需的指令或数据在片内coche中时， 消除了对总线的访问。 因为与总线长度相比 ， 处理器内部的数据路径较短， 访问片内也che甚至比零等待状态的总线周期还要快。而且，而且， 在这段时间内， 总线是空闲的， 可用于其他数据的传送。

片内cache导致了另一个问题：是否仍需要使用一个片外的或外部的coche。 通常 ， 答案是肯定的， 多数当代的设计既包含片内cache, 又包含外部coche。 这种组织方式中朵简单的是两级coche, 其中， 片内cache为第一级(LI), 外部coche为第二级(12) 。 包含L2 coche的理由如 下 如果没打巳coche并且处理器要求访问的地址不在LI coche中时 ， 则处理器必须通过总线访问DRAM或ROM存储器。 因为通常总线速度较慢且存怕器存取时间较长， 这就导致了较低的性能 一方面， 如果使用了l2 SRAM (静态RAM) ,.,he, 则经常缺失的信息能够很快被取来。如果SRAM的速度快到能与总线速度相匹配， 则数据能够用笭等待状态来存取， 这儿总线传输最快的一种类型。

当代多级cache设计的两个特点值得注意 第一， 对于片外12 cache, 许多设计都不是用系统总线作为l2cache和处理器之间的传送路径， 而是使用单独的数据路径、 以便减轻系统总线的负担。 第二， 随疗处理器部件持续缩小， 现在巳有许多处理器将12,邸he结合到处理器芯片上， 改善了性能。

若想使用l2'配he, 则取决千LI和l2中的命中率。 一些研究表明 ， 使用两级也che通常确实可以提高性能（参见[ AZIM92]、 [NOVl93]、 [ HAND98]) 。 然而， 多级cache的使用也使得关千cache 设计的所有问题都变得复杂， 包括cache容从 、 替换馆法和写策略等， 详见 [ HAND98]和[ PEIR99]。

图4.17给出了在不同也ch, 大小情况下两级妞,he性能的模拟研究结果[ GENU04] 。 图中假定两级c缸he都有相同的cache行大小， 井给出了不同情况下的总命中率。 也就是说， 如果所需的数据在LI妞,he中或l2"'"'中出现， 则算一次命中 。 图中屈示了不同Llmhe大小下l2 cache大小对总命中率的影响。 直到l2cache大小至少为u .. ,h, 大小的两倍时， 才对提高总命中率有明显的作用。 注意：当LI也,he为8KB时， 曲线最陡的点出现在l2cache为16KB时。 同祥 当LIcache为16KB时， 曲线最陡的点出现在l2mhe为32KB时。 在最陡的点出现之前，L2 cache对总臼,he性能几乎没多少影响。 需要l2 cache比LIcache大时． 才能使提高性能成为可能。 如果l2 cache与Llmhe有相同的行大小和容肚， 则其内容将或多或少与LIcache中相同。

随籽适用干妞,h, 的芯片面积的可用性的提庙， 大多数当代处理器已将l2 c,che移到处理器 芯片上， 井添加了一个L3'砒he。 最初， L3cache是越过外部总线来存取的。 而从近， 大多数处理器已从成到L3cache上。无论从哪种情况，加入L3cache使性能明显地得到提升（参见CURR11）此外，大型系统（例如IBM大型机zEnterprise系统）现在包含3个片上高速缓存级别和跨多个芯片共享的第四级高速缓存[CURR11]。

（习题）

第五章

5.1 半导体主存储器

在早期的计11:机中， 主存估祥中的随机存取存储器最通用的形式是使用一组环形的铁磁体阴 称为磁芯。 因此 ， 主存储器通常称为核'"• 这一术语沿用至今。 在磁芯存储器消失以前， 微电子技术已经出现了很久 ， 优势巳很明显。 目前 ， 几乎所有的主存储器都采用半导体芯片。 本节将讨论这一技术的关键方面。

5,1,1 组织

半导体存估器的基本元件是存恬位元。 虽然有各种电子技术可采用， 但所有的半导体存储 位元都具有某些相似的性质：

• 皇现两种松态（或半稳态）， 分别代表二进制的1和0,

• 能够写入信息（至少一次）来设灶状态，

（缺少）

5.1.2 DRAM和SRAM

个六将讨论的所存存储器类型都是随机存取的， 即通过编排的寻址逻辑 ， 存储器的单个字立接被存取。

农5-1列出了半导体存估器的主要类型。垃常用的是随机存取存储另(modom知~凶m叩正可，RAM)。当然，这是术由误用，因为表中列出的所有类型的存储群都是随机存取的。RAM的明显特征是，可以方便快捷地从存估器读取数据和向存恬器写入新数据，且读写操作都是通过使用电信号来完成的。（缺表）

RAM另一个明显特征是易失性。 RAM必须持续供电． 一且断电， 数据就会丢失。 因此 ， RAM仅能用于暂时存估。 计打机中使用的两种传统的RAM形式怂DRAM和SRAM。

，． 动态RAM

RAM技术分为动态和静态两类。 动态RAM (dya叩,c RAM. DRAM)利用电容充电来存恬数据， 位元中的电容有、 无电荷分别代表二进制的1或0。 因为电容器有漏电的自然趋势， 因此 动态RAM需要周期地充电刷新来维持数据的存恬。 动态一词就是指这种存储电荷丢失的趋势， 即使电说一立在供电。

图5-2, 是存储1位信息的单个位元的典型DRAM结构。 当要读出或写人该位元的位值时 ， 激励地址线。 品体忤像开关一样工作． 如果布电压施加到地址线上， 品体管导通． 如果无电压施 加到地址线上 ． 则品体管开路（无电流通过） 。

对于写橾作． 一个电压信另施加到位线上 高电斤代表I, 低电压代表0。 然后一个伯号施 加到地址线． 允许电荷传输到电容器。

对T-i实操作 ， 当地址线被选中时， 品体竹导通， 存估在电容上的电荷被送出到位线和读出放大器。 读出放大器将此电容电兀与一参考{1'[进行比较， 并确定位元保存的是逻辑1还姑逻辑0。 位元的读：1: 放掉了电容上的电荷 必须犹新存储才打完成本次操作。

虽然ORAM位元能用来存储单一位伯(0或I), 但它本质上是一个模拟设备。 因为电容能存估一定范围内的任何电荷伯， 因此必须使用一个阔伯来确定该电荷仇代表的是＼还是0。

1. 静态RAM

相对而言 静态RAM(s,,r;, KAM, SRAM) 是一个数字设备， 它使用与处理器相同的逻铅 元件。 静态RAM采用传统的触发器、 逻钳门配出来存估二进制伯（诮见第20货中有关触发抖的描述） 。 只要屯说不断， SRAM将一虹保持它所存忧的数据。

图5-2b是单个位元的典型SRAM结构。 4个品体管(T,、 T,, T, 和T,)交叉连接组成一个 布铅定逻相状态的排列。 在逻钳状态I下 ， C, 点足窃电平而 c, 点是低电平． 此时 ， 品体管T, 和 兀截止， 而T, 和T, 导通°。 在逻轨状态0下 ， C, 点是低电平而 c, 点是抚电平， 此时， 品体管 T, 和T, 导通， 而T, 和T, 载止。 只要比流电税一直供电． 这两个状态就都是稳定的。 不同于ORAM, 这m不需要刷灶米维持数据。

如同ORAM中一祥， Sil从1地址线用来控制开关的通断。 这里 ， 地址线控制两个品体管(T, 和飞） ． 当信句·施加到地址线上时， 两个品体忤导通， 允许读／写操作。 对于写操作， 位优施加 到B线 ， 位仇的反施加到B线， 这强迫4个品体竹(T, 、 T,, T, 和T,)进人一个相应的稳态。 对于读操作， 位值由B线读出。

3. SAAM与DRAM对比

静态与动态RAM都是易失的． 即二者都要求电游持续供电才能保存位仇。 与静态存储器位元相比 ， 动态存估器位元要小， 而且电路更简单。 因此， 与SRAM相比， DRAM的密度要il'1; (较小的位元＝每单位面积上更多的位元） ， 且价格更使宜。 另一方面 ， DRAM要求有支持刷新的电路。 但是， 对于较大容拭的存储器， DR从1位元较低的可变成本足以补偿刷新电路的固定成本。 因此， ORAM更趋向于满足大容扭存估器的需求。 蚊后还需指出 ， 通岱SRAM要比ORAM快。 由千这些相对特征， SRAM一般用于"'"'"存估器（片上的或片外的） ， 而DRAM则用千主存估器。

5.1.3 ROM类型

顾名思义， 只读存储器(read-oaly memo,y, ROM)含有不能改变的永久性数据。 ROM是非

易失性存储器， 即存估器中的数据并不要求供电来维持。 ROM可读， 但不能写入新数据。 ROM的一个须要应用是饭程疗设计 ． 这将在第四部分讨论。 其他可能的应用包括，常用功能的了程序库 ， 系统程序 ， 函数表。

写过程是用电信号执行， 由供应面或用户在芯片出厂后写人一次。 f店要特殊设备来完成写或 “编程过程 。 PROM提供了灵活性和方便性， 而ROM在大批lit少产领域仍具右吸引力。

只读存储器的另一种变体处主要进行读操作(􀀊a,l"m􀀌tly)的存储器， 常用于读操作远多于 写扑作且要求非易失数据的应 用场合 。 常见的主要进行读操作的存储器有3种 EPROM, EEP· ROM和快闪存储器。

典型 的光可擦除／可编程只读存储器(,啦obi, pro严mm,bl, re;od-m,ly memo,y, EPROM)与 PROM一样可读 可写。 然而， 在写人操作前， 必须通过让封装芯片暴赏在紫外线辐射下使所存的 存储位元都被擦除 ， 以还原成初始状态。 拱除需要通过让设计在芯片上的协口在强紫外线下长

时间照射来完成。 这种报除过程 可狱复进行， 每次撩除需要约20分钟。 因此. EPROM可以修改

多次并且和ROM、 PROM一样能够长久保存数据。 对于等容lit的存估器来说. EPROM比 PROM更贤 ， 但它具有可多次改写的优点 。

更具吸引力的主要进行读操作的存忧器形式是电可撩除／可编程只读存储器(d􀁄trically

e啦,bl,p哼沺血诅bl, ,,.a.,,,\y memo,y, EEPROM)。 这种存储器在任何时候都可写人、 而无仙掠

除原先内容， 且只更斩寻址到的 一个或多个字节。 写橾作比汝操作时间要长得多， 每字节斋要几百微秒 的时间。 EEPROM把非易失性和数据修改及活的优点结合起来， 修改数据时只需要使用岱规的 控制、地址和数据总线。 EEPROM比EPI<OM贵 ， 且密度低， 支持小容扯芯片。

另一种半导体存储器是快闪存储器（伽sh m,m叩􀁪）． 这 是由于其重编程速度快而得名 。 快闪 存忧器在20世纪80年代中期首次推出 ， 其价格和功能介于EPROM和EEPROM之间。 与EEP· ROM相似， 快闪存忧器使用电撩除技术， 整个 快闪存储器可以在一秒或儿秒钟内被掠除 ． 速度

比EPROM快得多 。 另外， 它能擦除存储器中的某些块 ， 而不怂整块芯片。 快闪存储器用于饮芯 片中， 一次或 “一瞬间” 可以只探除 一部分存储器位元， 因此而ill名。 然而快闪存储器不提供字节级的 擦除 。 像EPROM一样， 快闪存估器每位只使用 一个品体管， 因此． 与EEPROM相比 较 ， 能获得与EPROM一样的环存估 密度。

5.1.4 芯片逻辑

如同其他染成电路产品 ， 半导体存估器也灿封装 的芯片（如图2,7所示） 。 每块芯片包含一组存储位元阵列。在整个存储器层次结构中， 需要在速度、 容员和价格之间进行权衡。 当我们若虑芯片的存估

位元组织 和功能逻辑时， 也要做这些 权衡。 对于半导体存储器． 一个关键的设计问题是每次可以

读／写数据的位数 。 一种极瑞的情况是阵列中位元的物理排列与存估器中字的逻辑排列（从处理 器的If!度行）相同 ， 阵列组织成W个字， 每个字B位。 例如. 16Mb的芯片能够组织成IM的16位字。 另一种极端的情况是所谓的 每芯片一位的结构， 此时数据每次只能读／写1位。 可以用 一个DRAM来说明存忧器芯片的结构， ROM的结构与之类似， 且更简单。

图5.3表示了16Mh DRAM的 一种典型结构。 这种情况下． 一次读或写4位。 逻机上， 存储 器组织成4个2048 X 2048 的方阵 。 可以采用各种物理排列。 在任一种情况下， 阵列元紫由行(row)控制线和列（创,mo)控制线连接， 每根行控制线迕接到它所在行中每个位元的Select端口 ， 而每根列控制线连接到相应列中每个位元的Data-In/Seo沈t.;口。

地址线提供了被选择字的地址， 总共需要log,W条线。 在这个例子中， 符要11根地址线来 选中2048行中的 一行． 这11根地址线连接到行译码器的输入线。 行译码器布11根输入线和 2048根输出线． 其逻钳依据11根输入线的位模式激活2048根输出线中的 一根(2" 02048 。)

另外的 11根地址线可选中2048列中的 一列， 每列巾4位组成。 4根数据线用于与数据缓冲 器交换4位数据。 愉人（写）时， 每根位线的位驱动器根据相应数据线的仇被激活为1或o, 输出（读 ）时． 每根位线的伯经过读 出放大器． 传递到数据线Jc。 行线选择哪一行的位元参与读或写操作。

因为此DRAM每次只有4位位元参与读／写， 因此 ， 必须将多片DRAM连接到DRAM控制器上才能读／写一个字到总线上。

注意 ， II根地址线(AO-AIO)只是选中2048x2048忤列所需地址位数的一半 ， 这样做是为了节省引脚数。 22根需要的地址线通过外部的选择逻辑连接到芯片， 并且被复用为II根地址线 首先， II位地址信号经过芯片去定义阵列的行地址， 然后， 另II位地址信号去定义列地址。伴随着这些信号打行地址志通（石AS)信号和列地址选通（石is)信号， 为芯片提供时序控制信号。

写允许(WEl和输出允许（玩）引脚确定完成的是写或是读橾作。 图5-3中未示出的另外两个引脚是地(V一 ）和电jffi(V.)。

此外， 地址线的复用和方阵型行列结构， 导致每出现新一代存储器芯片， 其容拭就以4倍的方式增长。 每增加一个专用的地址引脚， 便使行地址和列地址的指示范围加倍， 因此存储器芯片的容社以4 的倍数增长。

图5-3中还包含了刷新电路， 所有的DRAM都需要刷新操作。 实际上， 简单的刷新技术是使DRAM芯片丧失读写能力而刷新所有数据位元。 刷新计数器迫历通过所有的行值。 对每一行 ， 刷新计数器的值被当作行地址输出到行译码器， 井且激活RAS线， 数据被读出后又写回原地址， 从而使得相应行的所有位元被刷新。