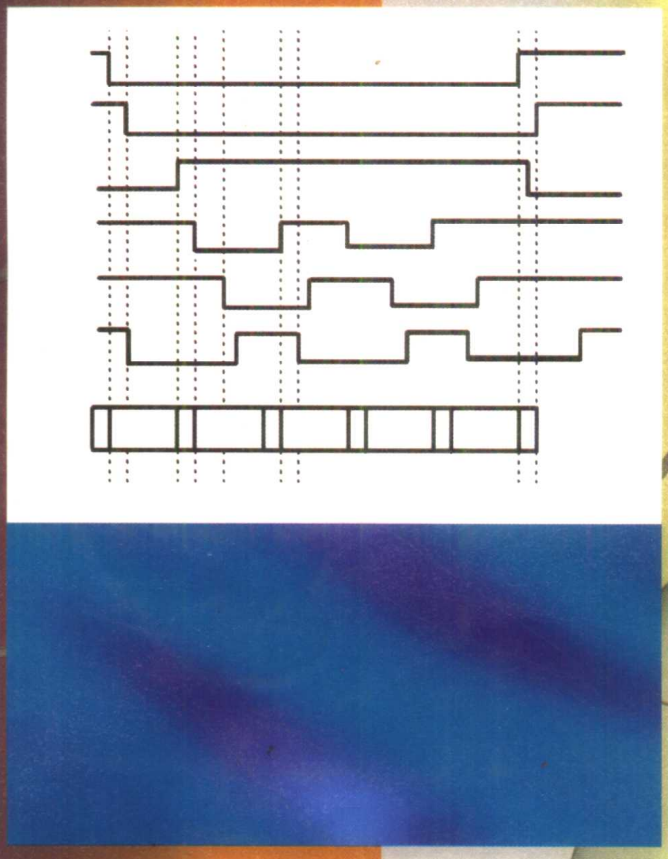
高级计算机结构技术及其应用

苏广川　张笈　编



北京理工大学出版社

高级计算机结构技术及其应用

苏广川　张　笈　编

北京理工大学出版社

内 容 简 介

本书对现代计算机的典型结构(流水线计算机、向量计算机、数据流计算机和多处理机)、Pentium 高级微处理器的内部组成、虚拟地址保护方式的工作原理和编程方法、先进的系统总线结构(PCI、Futurebus+、SCSI)和现代存储器结构(Cache 存储器、高级 DRAM、磁盘阵列和 CD-ROM 等)都作了系统、深入的介绍。并结合当代计算机技术、网络通信技术和信息处理技术相互渗透的特点,对多媒体网络技术和高速数字信号处理(DSP)技术也作了较全面的阐述。

本书内容丰富、技术性强,具有较高的理论性、系统性和实用性,可供研究生、本科生高级班的计算机教学使用,亦可作为计算机工程专业人员的参考资料。

图书在版编目(CIP)数据

高级计算机结构技术及其应用/苏广川,张笈编. —北京:北京理工大学出版社,1998.11
ISBN 7-81045-497-8

Ⅰ.高… Ⅱ.①苏… ②张… Ⅲ.电子计算机-系统结构 Ⅳ.TP303

中国版本图书馆 CIP 数据核字(98)第 29886 号

责任印制:刘季昌 责任校对:陈玉梅

北京理工大学出版社出版发行
(北京市海淀区白石桥路 7 号)
邮政编码 100081 电话(010) 68912824
各地新华书店经售
北京地质印刷厂印刷

*

787 毫米×1092 毫米 16 开本 19.75 印张 484 千字
1998 年 11 月第 1 版 1998 年 11 月第 1 次印刷
印数:1—1500 册 定价: 30.00 元

前　言

当代计算机科学技术的飞速发展，对国家的经济腾飞、社会生活乃至人类文明都起着不可估量的影响。计算机技术、多媒体网络技术、信号处理技术的同步发展，促进了信息技术与产业的新的革命，致使社会生产力发生深刻的变革和人们的生活方式产生巨变，本书正是围绕上述三个方面的有关技术问题进行探讨。

纵观计算机发展历史可以得出这样一条结论，即计算机性能的不断提高，不仅依赖于计算机器件的变革，同时还需依靠计算机系统结构的改进。本书着重介绍现代计算机所采用的一些先进结构以及这些先进技术如何在诸如 Pentium 等高级微处理器中的具体运用；对现代计算机所采用的总线结构（如 PCI、Futurebus＋、SCSI）和现代存储器结构（如 Cache 存储器、高级 DRAM、磁盘阵列、CD－ROM 等）也作了系统、深入的介绍。结合当代计算机技术、网络通信技术和信号处理技术的相互渗透、相互促进的特点，本书对多媒体网络技术和高速数字信号处理（DSP）技术也作了较全面的阐述。

本书内容丰富、技术新颖、图文并茂、深入浅出，具有较高的理论性、系统性和实用性，适用于从事机、电等专业的研究生计算机教学用书，同时也是为工程硕士班编写的计算机教学参考书。

全书共分八章：第一章计算机结构体系概述，介绍流水线计算机、向量计算机、数据流计算机和多处理机的结构与设计技术；第二章高级微型计算机系统结构，重点介绍 Pentium 微处理器的内部结构、组成和技术特点；第三章高级微处理器的操作方式，介绍 32 位微处理器的实地址工作方式、保护方式和虚拟 8086 方式，侧重讨论虚拟地址保护方式的工作原理及如何支持多任务和多用户操作；第四章指令系统及其编程，重点介绍 32 位机所特有的指令系统类型、功能及保护方式下的汇编语言程序设计方法；第五章系统总线结构，介绍为高级微机系统广泛采用的 PCI 总线、SCSI 等总线结构特点；第六章现代存储器结构，主要结合多媒体计算机中采用的 Cache 存储器结构、CD－ROM 等光存储器、并行计算机中采用的新型 DRAM 结构和网络服务器中的磁盘阵列技术。第七章多媒体网络技术，在介绍网络常用技术的基础上，重点阐述支持多媒体网络的 ATM 技术特点；第八章高速数字信号处理器（DSP）及其应用，介绍典型的 DSP 的技术特点，TMS320 C3X 的软、硬件以及在数字信号处理中的应用。

在编写这本研究生教材的过程中，得到我校研究生院负责人冯淑华教授的大力支持和指教，对此表示衷心地感谢。

由于时间仓促和水平的限制，书中会有不少的缺点和问题，敬请读者批评指正。

编　者

1998.2.10

目　录

第一章　计算机结构体系概述

§1.1　计算机结构概述

近半个世纪以来，计算机在硬件和软件方面都以极其惊人的速度飞跃发展。它对现代科学技术的发展产生了巨大的推动作用，对人类的经济、社会生活乃至人类的文明起着不可估量的作用。近十年来，计算机、通信技术的同步发展，促进了信息技术与产业的新的革命，信息革命必将使社会生产力发生深刻的变革，使人们的社会生活发生巨变，而现代社会的需求也反过来促使计算体系结构的重大革新。

自计算机诞生以来，大致经历了以下几个发展阶段：

第一代计算机(1945—1954)为电子管时代，它由单个中央处理器(CPU)组成。CPU 用程序计数器、转移指令和累加器顺序完成定点运算，并对存储器和输入/输出设备进行访问。计算机采用的是机器语言或汇编语言。

第二代计算机(1955—1964)为晶体管时代，它采用了变址寄存器、浮点运算、多路存储器和 I/O 处理机，并采用了 FORTRAN、ALGOL、COBOL 等高级语言以及编译器和监控程序等。

第三代计算机(1965—1974)在逻辑电路和存储器方面采用了中、小规模集成，采用了微程序控制。为缩小 CPU 和主存储器之间的速度差异，采用了流水线和高速缓存，为使 CPU 和 I/O操作在多用户程序之间交叉进行，还采用了进程管理，同时也促进了有虚拟存储器的分时操作系统(OS)的发展，使资源多路切换得到共享。

第四代计算机(1974—1991)采用的是大规模和超大规模集成电路，出现了共享存储器、分布存储器或向量硬件选件的不同结构的并行计算机，开发了用于并行处理的多处理机操作系统、语言、编译器和环境。

第五代计算机(1991 至今)为人工智能机，它将是具有渊博的知识、能推理、会学习的计算机。第五代计算机应具有一个和谐的人机环境，使人们认识客观问题时的认识空间与计算机处理问题时的处理空间尽可能地一致。为此需解决人头脑中属于并发的、联想的、形象的和模糊思维，翻译成为计算机善于接受的程序，这就需要计算机具有极大规模并行处理能力，寻找一种通用的、能适应各种应用问题的极大规模并行算法。

从信息处理的类型来看，无论是自然信息还是人类活动的信息，有 90%以上都是非数值信息。这些非数值信息的处理基础主要是代数运算，即运用布尔代数、集合代数、关系代数和命题代数等规则进行操作。实际上，客观世界的信息模式是多样、相互依赖、不断变化的，并受某种条件约束且遵从一定的逻辑规则，这就要求计算能获取多媒体信息，具有高性能信息加工装置，向人类提供声、图、文集成在一起，并能与人类发生动态交互作用的信息，具有知识库及管理系统，能够学习、联想和推理，尽可能地以接近人的思维方式去认识处理问题，藉此以提高人与信息社会的接口能力，提高人对信息的理解能力，更进一步发挥信息的共享性和增值性。因此第五代计算机必须在系统结构、软件和工艺上进行开创性的研究，并进一步开拓计算机的新

的应用领域。

回顾计算机的发展历史可以看出，计算机系统性能不断提高，不仅要依靠器件的变革，还需要依靠计算机系统结构的改进。所谓计算机系统结构是指把各个功能部件组成一个系统，这些部件可以是硬件、软件或两者的混合体。系统结构设计是选择一种最佳的部件组合，使得整个系统有效地工作。因而计算机系统结构的设计是在功能这一层次上考虑问题。计算机组成是计算机系统结构的逻辑实现，按照所希望达到的性能价格比，最佳、最合理地把各种设备和部件组成计算机，以实现所确定的计算机结构。

现代计算机系统结构与冯·诺依曼(VON·NEUMAMM)等人当时提出的计算机系统结构相比，虽已发生重大变化，但就其结构原理来说占有主流地位的仍是以存储器原理为基础的冯·诺依曼型计算机。存储程序原理的基本点是指令驱动，即程序由指令组成，并和数据一起存放在计算机存储器中，机器一经启动，就能按照程序指定的逻辑顺序把指令从存储器中读出来逐条执行，自动完成由程序所描述的处理工作。但近四十多年来计算机的系统结构也进行了许多重大改进，主要表现在以下几个方面：

① 计算机的系统结构从基于串行算法改变为适应并行算法，从而出现了向量计算机、并行计算机和多处理机。

② 计算机系统结构从传统的指令驱动型改变为数据驱动型和需求驱动型，从而出现了数据流机器。

③ 计算机系统与通信网络紧密结合，从而出现了各种功能分布计算机。

④ 出现了处理非数值化信息的智能计算机。例如自然语言、声音、图形和图像处理等，它主要依靠有关知识进行逻辑推理，特别是利用经验性知识，对不完全确定的事实进行模糊推理。

计算机系统结构可依据不同的特征进行多种分类，这里主要介绍以指令流和数据流的不同的组织形式进行分类的 Flynn 分类法。

1966 年 M·J·Flynn 提出了定义：

指令流(Instruction Stream)——机器执行的指令序列。

数据流(Data Stream)——由指令流调用的数据序列，包括输入数据和中间结果。

多倍性(Multiplicity)——在系统最受限制的元件上同时处理同一执行阶段的指令或数据的最大可能个数。

按照指令流和数据流的不同组织形式，把计算机系统的结构分为以下四类：

① 单指令流、单数据流 SISD(Single Instruction Stream Single Data Stream)；

② 单指令流、多数据流 SIMD(Single Instruction Stream Multiple Data Stream)；

③ 多指令流、单数据流 MISD(Multiple Instruction Stream Single Data Stream)；

④ 多指令流、多数据流 MIMD(Multiple Instruction Multiple Data Stream)。

对应于这四类计算机的基本结构框图如图 1-1 所示。

其中图(a)为传统的顺序执行的单指令流、单数据流 SISD 计算机。图(b)为向量计算机结构，以单指令流、多数据流 SIMD 形式出现。图(c)为多指令、多数据流 MIMD 的并行计算机。图(d)为多指令流、单数据流 MISD 的流水线的搏动阵列(Systolic arrays)。

四种机器模型中，过去制造的大部分并行计算机都采用了适合于通用计算的 MIMD 模型。SIMD 和 MISD 模型更适合于专用计算。

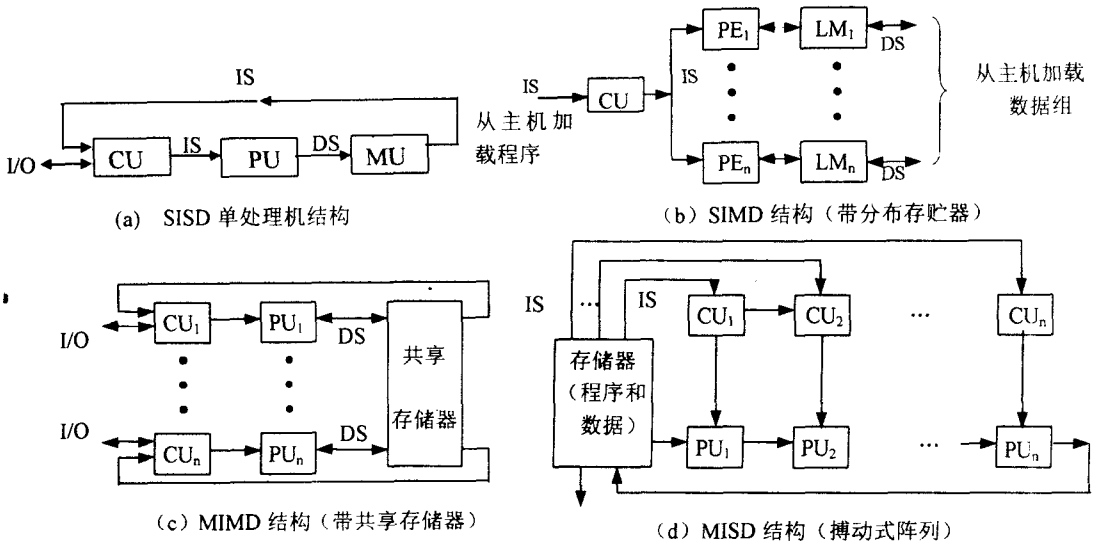


图 1-1 计算机系统结构的 Flynn 分类

CU—控制部件；PU—处理部件；MU—存储部件；IS—指令流；DS—数据流；PE—处理单元；LM—本地存储器

§1.2 流水线计算机设计技术

流水线技术早已应用于社会生产与活动的各个方面，其主要特点是在老任务完成之前可以开始一个新任务。被处理的数据在流水线的起点连续送入，计算结果在流水线终端不断送出，完成任务的速度只取决于能够提供新任务的速度，而与任务所需的全部处理时间无关。流水线计算机技术是通过并行硬件来改善系统性能的最普通的手段，是缓解计算机瓶颈的又一途径。

根据对数据流沿流水线流动的控制方式，线性流水线可分为异步和同步两种模型。

在异步流水线中，相邻段之间的数据流是由一个应答(hand shaking)协议来控制的。如图 1-2(a)所示。当 S_i 段准备传送时，给 S_{i+1} 段发送一个就绪信号，S_{i+1} 段接到输入数据后，回送一个应答信号给 S_i 段。异步流水线常用于消息传递型多计算机系统的通信通道设计中，异步流水线的吞吐率是可变的，不同段可以有不同的延迟量。

同步流水线模型如图 1-2(b)所示。图中的时钟锁存器作各段之间的接口，它可用主-从触发器实现，将输入和输出隔离开，当时钟脉冲到达时，所有的锁存器同时将数据传送到下一段。流水线段由组合逻辑线路构成并要求各段的时延大致相等。这些时延决定了时钟周期，从而也就决定了流水线的速度。

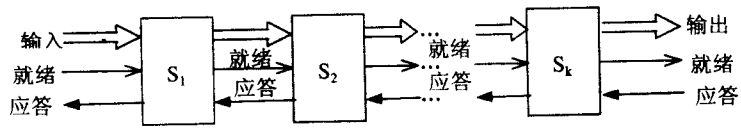
流水线的时钟周期 τ 可按(1-1)式确定

$$\tau = \max_i\{\tau_i\} + d = \tau_m + d \qquad (1-1)$$

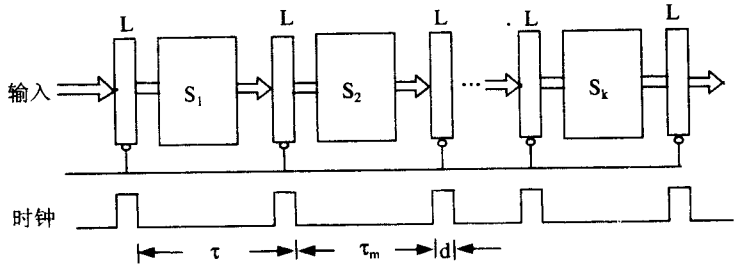
其中：τ_i 为 S_i 段线路的时间延迟，d 为锁存器的时间延迟，τ_m 为最大段延迟。

如果每个周期能从流水线送出一个结果，则流水线的最大吞吐率 f 为

$$f = \frac{1}{\tau} \qquad (1-2)$$



（a）异步流水线模型



（b）同步流水线模型

图 1-2　线性流水线部件的两种模型

S_i—段 i；L—锁存器；τ—时钟周期；τ_m—最大段延迟；d—锁存器延迟

在理想情况下，一条 k 段线性流水线能在 $k+(n-1)$ 个周期内处理 n 个任务。其中 k 个周期用来完成第一个任务的执行，其余 $n-1$ 个任务则需要用 $n-1$ 个周期，因此要求总的时间为

$$T_k = [k + (n-1)]\tau \tag{1-3}$$

其中 τ 为时钟周期。假如一台功能上等效的非流水线处理机的流动延迟为 $k\tau$，则在此非流水线处理机上执行 n 个任务需要的时间为 $T_1 = nk\tau$。

一条 k 段流水线对一台等效的非流水线处理机的加速因子可定义为

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} = \frac{nk}{k + (n-1)} \tag{1-4}$$

可见当流水线执行的任务（操作或指令）数 $n \to \infty$ 时，最大加速比即为 $S_k \to k$。这种最大加速比是很难达到的，这是因为相继任务（指令）之间存在着数据相关、程序分支等一些因素的缘故。当流水线的分段数 k 较大时，可以得到加速性能的提高。但由于实际成本、控制的复杂性、电路的实现及组装技术等问题的限制，流水线的段数也不能无限制地增加。实际上，大多数流水线在功能上分段，段数的范围为 $2 \leqslant k \leqslant 15$。在实际计算机中，极少有流水线设计成超过 10 段的。流水线段数的最佳选择能使性能价格比达到最高。

假定一个给定功能的非流水线顺序程序所要求的总执行时间为 t，如果在一个有相等流过延迟 t 的 k 段流水线上执行同一个程序，则需要的时钟周期为 $P = t/k + d$，其中 d 为锁存器延迟。这样流水线的最大吞吐率为 $f = 1/p = 1/(t/k+d)$。流水线的总价格可粗略估计为 $c+kh$，其中 c 为所有逻辑段的价格，h 为每个锁存器的价格，流水线的性能价格比（PCR）的定义为

$$PCR = \frac{f}{c + kh} = \frac{1}{(t/k + d)(c + kh)} \tag{1-5}$$

流水线段数的最佳值为：

$$K_0 = \sqrt{\frac{t \cdot c}{d \cdot h}} \tag{1-6}$$

其中 t 是流水线的总流过延迟。

一个 k 段线性流水线的效率 E_k 可定义为

$$E_k = \frac{S_k}{k} = \frac{n}{k + (n-1)} \tag{1-7}$$

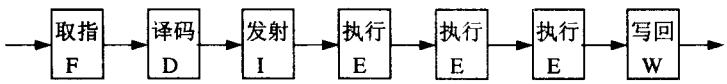
很明显，当 n→∞时，效率接近于 1，在 n=1 时，E_k 即为 $1/k$，是其下限。流水线的吞吐率 H_k 则可定义为每单位时间内执行的任务数：

$$H_k = \frac{n}{[k + (n-1)]\tau} = \frac{nf}{k + (n-1)} \tag{1-8}$$

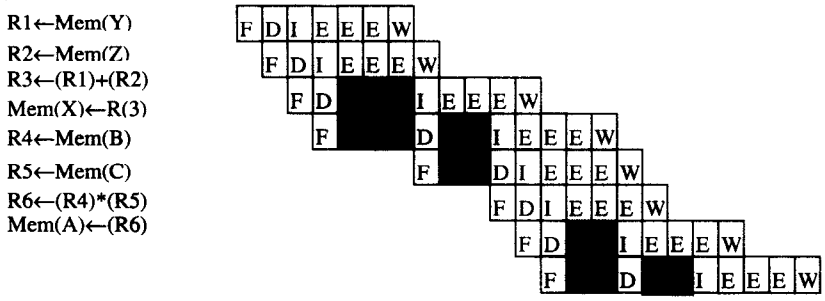
当 $n \to \infty$，$E_k \to 1$ 时，就得到最大吞吐率 f。

指令流可以用流水线以重叠方式执行。一条典型指令的执行由一系列操作组成，包括取指令、译码、取操作数、执行以及写回几个阶段。理想情况下，这些阶段在线性流水线上可重叠执行。每一个阶段的执行可以用一个或几个时钟周期，这取决于指令的类型以及所用处理机/存储器的体系结构。

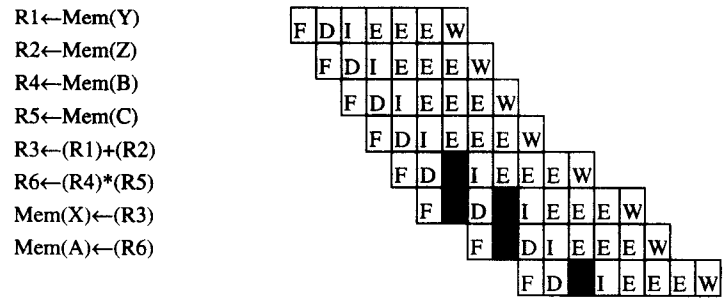
一条典型的指令流水线如图 1-3 所示。



（a）一条 7 段指令流水线



（b）按次序发射指令



（c）重新排序后发射指令

图 1-3　X=Y+Z 和 A=B×C 的流水线执行

取指段(F)从高速缓存取指令，译码段(D)译出所要执行指令的功能并识别出所需要的资源。这些资源包括通用寄存器、总线和功能部件。发射段(Issue Stage)(I)保留资源，提供流水线控制互锁，在发射段期间还可以从寄存器读出操作数。

指令在一个或几个执行段(E)中执行，图 1－3(a)中有三个执行段，存储器的取数或存数是执行的一个部分。最后的写回段(W)用来将结果写入寄存器。图中 8 条指令反映的是流水线在执行高级语言语句 X＝Y＋Z 和 A＝B×C。假设执行取数和存数指令需要 4 个时钟周期，而浮点加法和乘法操作需要 3 个时钟周期。

图 1－3(b)说明指令按原来程序的次序发射的情况，阴影格子对应的是空闲周期，这是因为资源等待、资源冲突或因数据相关而使指令发射产生了阻塞。前二条取数指令是在接连的周期内发射的，而加法则要依赖于两次取数，因此在数据(Y 和 Z)取得之前一定要等待三个周期。

类似地，由于流相关，也必须等待三个周期。以便加法完成后才能把和存入存储单元。在 A 的计算期间也有类似的阻塞情况。这样，按序发射指令需要的总时间为 17 个时钟周期。这一时间的测量是从周期 4 开始，当第一条指令开始执行时算起，直到周期 20 最后一条指令开始执行时为止。这样的时间测量办法消除了流水线的“启动”或“排空”延迟的不适当的影响。

为了消除因相关性引起不必要的延迟，可对指令的发射次序做修改，得到改进的时间图表示于图 1－3(c)。其办法是将所有四条取数操作指令在一开始就发射，由于这些数据的预取，因而加法和乘法指令被阻塞的周期数都比较少，使计算时间减少到 17 个周期。

流水线设计方法是要制造一个简单的控制器，用它来控制流水线入口处的各种操作。控制器在入口处不断接收新操作进入流水线，使流水线一直维持在最大吞吐率，同时又要确保在流水线的各段中不会有两个或多个操作相互冲突，甚至当不同的操作按不同的路径通过系统时也不应有冲突发生。

这里以浮点算术部件为例来讨论使流水线具有最大吞吐率时需要的控制方法。

设浮点加法器与乘法器组合成一个流水线的结构如图 1－4 所示。

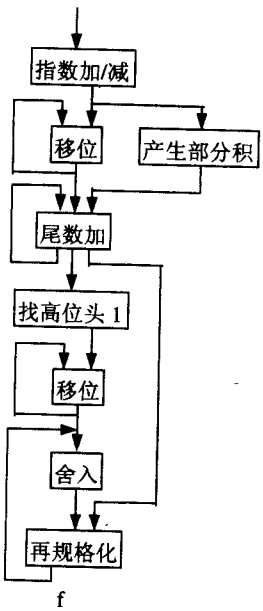
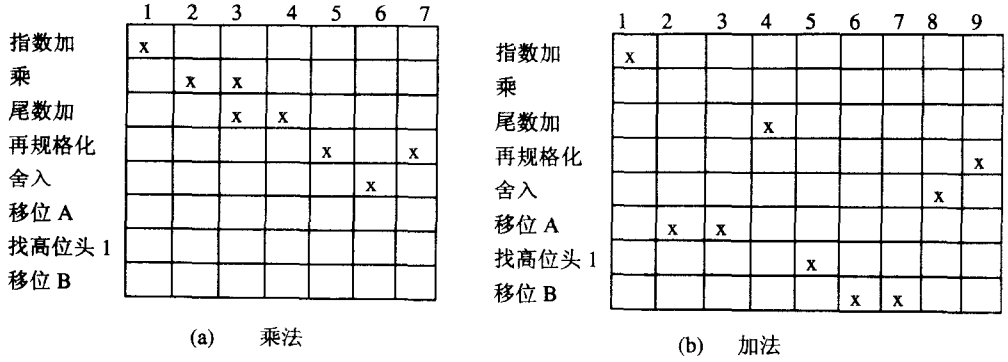


图 1－4　浮点加法器与乘法器组合成一个流水线的结构

为了预测冲突，需要研究一下数据流通过浮点算术部件的时间图即预留表，表中给出所需的时间信息。图 1－5(a)表示浮点乘法操作，表中的每一行表示算术部件内的各段，每一列表示一个时间步。

在图 1－5(a)中假定一个运算连续通过前两步到达执行尾数相乘的段，为了便于讨论，还假设尾数乘需要两个时间步来形成各部分积，两个时间步用来加部分积，最后三步是把乘积规格化，舍入成单字长。如果尾数溢出再规格化。

浮点加法的预留表如图 1－5(b)所示。可以看出该表中的数据流与浮点乘法中的数据流差别甚大，但是该表中的各行与图 1－5(a)中的各行段名相同、顺序相同，问题是如何利用这个预留表。在预留表中已经包括了研究控制方法所需要的全部信息。

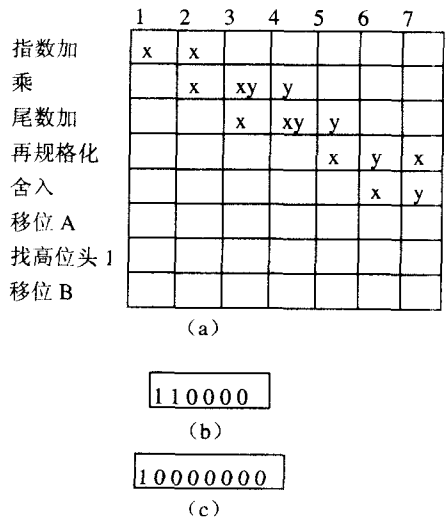


(a) 乘法　　(b) 加法

图 1-5　浮点运算部件的预留表

预留表是从流水线设计中直接推导出来的，对于单一功能的流水线，为了推导出控制算法，可将预留表做成一个模板，把它覆盖在另一相同的预留表上，用移动模板表示启动一个操作。当把模板相对于另一个预留表移动一个时间单位时，就会出现图 1-6(a)所示的样子。X 是原图中的标记，Y 是模板中的 X 右移一个时间单位后的位置。可以看出在有 X 或 Y 标记的方格表示在相应的时间相对应的功能处于忙状态，并表明该功能部件已被预留。如果方格中同时含有 X 和 Y，表示两个乘法操作同时需要一个功能部件，这就发生了冲突。

在图 1-6(a)中乘法部件在第三个时钟有一个冲突，由此可以得出结论，在发生第一个乘法操作后一个时间单位内不能发送另一个新的乘法操作。图 1-6(b)表示用二进制向量来描述冲突信息，该二进制向量就称为冲突向量。某一位为 1 表示将发生冲突，为 0 表示不会发生冲突。图 1-6(b)的整个冲突向量是 110000，表示在启动一个乘法后紧跟着两个时钟周期内，如果送入一个新的乘法操作，就会发生冲突。但在两个时钟周期以后的任何时间都可以启动另一个新操作。



(a)　(b)　(c)

图 1-6　乘法冲突向量的引出

加法的冲突向量如图 1-6(c)所示，其值为 10000000，说明在加法操作后的一个周期不能启动新的加法操作，而后七个周期中任何一个都可以开始新的加法操作。从预留表中可以看出流水线中有两个移位器将有助于消除加法器中的冲突。利用共享逻辑电路虽然可以节省硬件，但是限制了启动新操作的频率。

对于单一功能流水线控制器就像一个移位寄存器，它在每个时钟周期将其内容向左移，控制器的操作过程如下：

① 如果在某个时钟周期内从移位寄存器移出的是 0，则允许访问流水线的请求；如果是 1，则拒绝现行周期访问，并把请求保持到下个周期。

② 如果允许请求，就在现行周期结束时修改移位寄存器，它的新内容是它移位后的值与冲突向量的“或”。

③ 如果不允许请求，修改寄存器的方法是将其内容左移，从寄存器右端移入 0。如果出现请求时只有一个操作正在进行，移位寄存器的冲突向量移位后的内容就指明了流水线为避免冲突在启动之间所需要的相对延迟。如果两个或多个操作正在进行，并且新操作可能和任一操作发生冲突，这个新操作就不能开始。

分析分解状态图中的环路，可以发现流水线能够维持的最大操作速度的控制策略。图 1-7 为冲突向量 1001011 的分解状态图，图中的状态是刚刚发出一个新操作的各种可能的状态，箭头上的标号表明两状态之间的延迟时间。

建立分解状态图的算法概括如下：

① 先建立只有一个节点的初始状态。初始状态中放入冲突向量，它表示一个操作刚刚向流水线发送时控制器的状态。

② 检查初始状态中的各位，对其中的每个 0 都要找出：当这个 0 从移位寄存器中移出时，向流水线发送一个新的操作，控制器的移位寄存器应该得到的状态是相应于第 i 个 0 的状态。这个状态由当前状态左移 i 位，右端补 0，然后再与原冲突向量进行逻辑“或”运算。

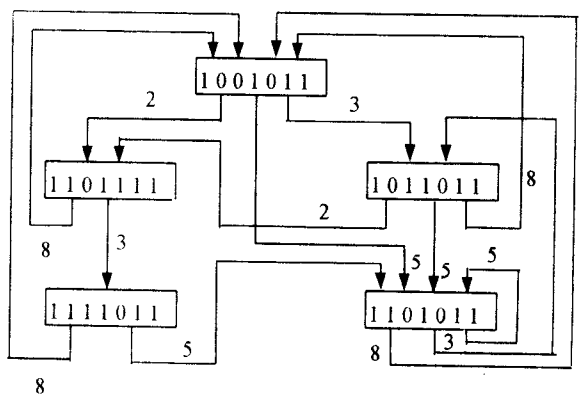


图 1-7　冲突向量 1001011 的分解状态图

③ 如果按上述方法得到的下一个状态是个新状态，则将它画入状态图中，并放入要检验的状态表。

④ 从状态图的当前状态到下一状态画一条箭头连线，并用 i 标记，i 表示从当前状态到下一状态的延迟周期数。

⑤ 从各个状态到初始状态都加一条箭头连线，并标记为 N。N 等于移位寄存器的位数加 1。

⑥ 当一个状态的下一个状态全部建立后，返回初始状态并检查有没有新状态要处理，如果没有，分解状态图就完成了。

分解状态图中的环路对应于重复启动操作的某种方法。例如从 1011011 到 1101011 的长度为 8 的环路，每 8 个周期完成两个操作，因此完成速度为每周期 0.25 个操作。

按控制策略应从延迟 2 开始，然后进入 3,5,3,2 环路，这是一个稳定的环路，每 13 个周期完成四个操作，完成速度是每周期 0.31 个操作。这一特殊的环路使流水线具有最高利用率。流水线的设计者通常应当完成上述分析方法，找出具有最高速度的环路，确定控制器的控制策略，以求实现最快的启动速度。作为对控制策略效率的一种测量，可以从预留表知道最大可能启动速度的大致范围。

影响流水线以可能的最大速度连续启动是因为流水线内发生了冲突，妨碍了新操作启动。如果在流水线中增加一些延迟线，往往可以获得最佳性能，虽然延迟线延长了通过流水线的总时间，但只要它们能排除限制启动速度的冲突就可以提高操作速度。

为了使取指令的速率和流水线消耗的速率相匹配，可以使用三种类型的缓冲器。在一次存储器访问的时间里，取出一组相继指令并放到预取缓冲器中，如图 1－8 所示。

成组访问可以用交叉存储模块来实现或采用高速缓存来缩短有效存储器访问时间。

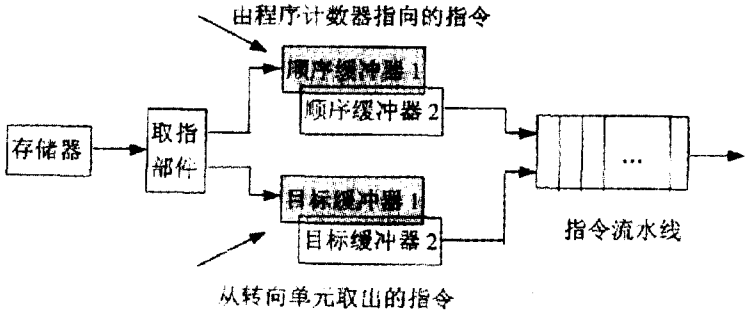


图 1－8　顺序和目标缓冲器的使用

对于顺序流水线，顺序指令装入一对顺序缓冲器。对于非顺序流水线，从转移目标取出的指令装入一对目标缓冲器。两种缓冲器都以先进先出方式操作，这些缓冲器作为附加段而变成流水线的一部分。

条件转移指令会使顺序缓冲器和目标缓冲器都装满指令。在转移条件被检测之后，从两个缓冲器中的一个取出合适的指令，而另一个缓冲器中的指令就被废弃。因此，人们可以用一个缓冲器装来自存储器的指令，而用另一个缓冲器把指令馈入流水线。这两个缓冲器轮流使用即可在指令流进和流出之间避免冲突。

另一类型的预取缓冲器称为循环缓冲器(Loop buffer)。这种缓冲器保存了一个小循环的顺序指令。循环缓冲器由流水线取指段提供指令，所预取在循环体内的指令将重复地执行，直到所有的循环迭代执行完为止。循环缓冲器的操作分两步：第一步它顺序地保存当前指令之前的指令，这就节省了从存储器取指令的时间；第二步它要识别出转移的目标何时落在循环的边界之内。在这种情况下。如果目标指令已经在缓冲器中，则可避免不必要的存储器访问。

此外，当某一个流水线段变为瓶颈时，它一定与预留表中最大符号个数的那一行相对应。这种瓶颈问题可以同时用多个同样的段来缓解，这就出现了在一台流水线处理机的设计中使用多个执行部件的情况，如图 1－9 所示。

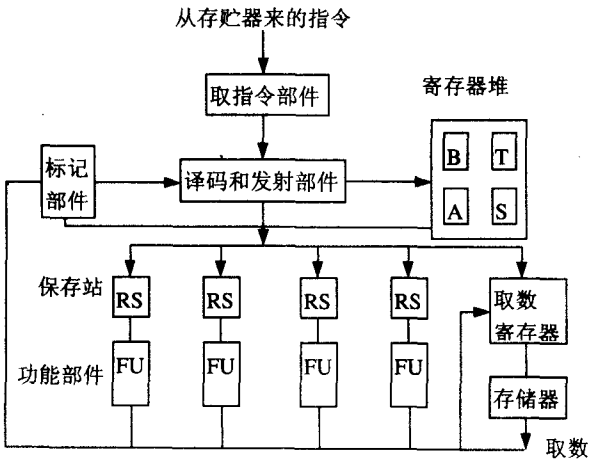


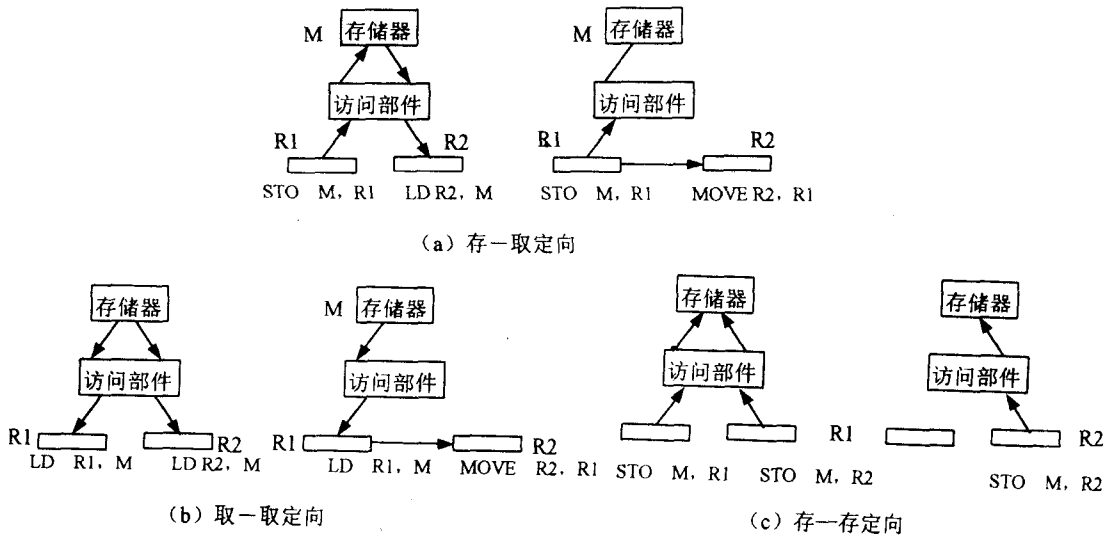
图 1－9　一台具有多功能部件和标记支持的分布式保存站流水线处理机

为了解决进入流水线相继指令间的数据或资源的相关性，需对每个功能部件使用保存站(Reservation Station，RS)。操作数可以在 RS 部件里等待，直到其相关问题解决为止。每个 RS 有一个标记可被唯一地识别，这些标记则由一个标记部件来监控。

标记部件始终要检查从所有当前使用的寄存器或 RS 来的标记。这种寄存器标记技术可以利用硬件来解决分配多条指令给目的和源寄存器之间的冲突。除解决冲突外，RS 也用作缓冲器，作为流水线的功能部件与译码、发射部件之间的接口。一旦解决了相关性的问题，多

功能部件就能并行地操作，这将缓解指令流水线执行段的瓶颈问题。

流水线处理机的吞吐率还可以用多个功能部件间的内部数据定向得到进一步改善。其基本思想是有些存储器访问操作能用寄存器传送操作取代，图 1－10 表示了这种传送过程。



（a）存一取定向

（b）取一取定向

（c）存一存定向

图 1－10　用寄存器传送操作取代存储器访问操作的内部数据定向

存-取定向(Store-load for warding)技术如图 1－10(a)所示，图中从存储器到寄存器 R2 的取操作(LD R2,M)可用从寄存器 R1 到寄存器 R2 的传送操作(MOVE R2,R1)来取代。由于寄存器传送比存储器访问要快，因此这种数据定向将减少与存储器的交往，从而缩短了执行时间。

类似地，用取—取定向(Load－load for warding)如图 1－10(b)所示，可取消第二次取操作(LD R2,M)，而用传送操作(MOVE R2,R1)来取代。存—存定向(Store-Store for warding)如图 1－10(c)所示，两个存操作一个紧接着另一个执行，这样，第二个存操作写在第一个存操作的上面，因而第一个存操作就变成多余的了，可将其删除而不影响结果。

流水线处理机的性能受到数据相关性及转移指令的限制。当发生转移时，流水线中跟在转移之后的指令就变成无用，需从流水线中排出，因而就损失了许多有用的周期。设计人员为减少条件转移对性能的影响可采用如下几个技术措施：① 延迟转移法；② 预测转移法；③ 按历史地址转移法。

延迟转移的目的就是要使转移的损失降至最少，这种思想原先是用在微指令编码中，如果延迟转移执行周期为 d，则跟在转移取之后至少有 $d-1$ 条有用的指令可以执行。这些指令的执行应该和转移指令的结果不相关，否则就不能使转移损失为 0。

用代码移入延迟时间片的延迟转移过程如下示代码：

⋮ ⋮

I1. Load R1,A

I2. DEC R3,1 I2. DEC R3,1

I3. Brzero R3 I5 I3. Brzero R3,I5

I4. Add R2,R4 I1. Load R1,A

I5. Sub R5,R6 I4. Add R2,R4

I6. Store R5,B I5. Sub R5,R6

原先的程序 I6. Store R5,B

把有用的指令移入延迟时间片

在转移不发生的情况下,执行修改后的程序和原先的程序得到同样的结果。在发生转移的情况下,修改后程序中经过延迟的指令 I1 和 I5 的执行都是有效指令。在大部分 RISC 处理机中包括 MIPS R4000、Motorola 的 MC68110 以及 TI 的 TMS320 C30 等处理器都已实现了延迟转移。

第二种方法是猜测转移地址并在流水线中按路径处理。猜选哪一个分支是固定的,可以是转移成功的分支,也可以是转移不成功的分支,因此可以使流水线按原来顺序继续向前流动。若猜错了,就要重新回复到分支点。控制机制的设计要保证在猜错而需返回到分支点时,能恢复分支点的原有现场。为了在猜错时能尽快地返回、转入另一分支,与沿猜测路径向前流动的同时,还可以由存储器预取猜错分支的头几条指令,存放在转移目标缓冲器,以便在猜错返回时,不必从访问内存取指令开始,这样就减少了延误返回的时间。

由于程序结构的特点,条件转移两个分支的出现概率是可预估的。如果程序设计者或编译程序能把出现概率高的分支安排为猜选分支,就会减少由于处理条件转移所引起的时间损失。

由于程序中广泛采用循环结构,流水机器多数采取特殊措施以加快循环程序的处理。各种措施的基本出发点,一是如何使整个循环程序都放入指令缓冲器,以减少执行循环程序时访问主存的次数;二是考虑对循环程序出口的分支处理。由于返回到循环程序本身的概率要比转移到另一程序的概率高得多,因此应尽可能使指令缓冲器的这种循环程序能首尾连接、连续流动。

为了把预测转移的性能改进得更好,就要使它能适应程序的执行特性,采用动态转移策略用近期的转移历史来预测下一次转移发生时是否要进行转移取。动态转移策略分为三种主要类型:第一类是以在译码段得到的信息为基础去预测转移方向。第二类是在计算转移目标的有效地址的那一段内用高速缓存来保存目标地址。第三类是在取指段用高速缓存保存目标指令。所有动态预测都将随程序的执行作动态地调整。动态调整需要附加硬件来保持对转移指令在运行对过去行为的跟踪,历史的记录量应可能少,否则,预测逻辑代价太高,以致很难实现。

§1.3 向量计算机设计技术

向量是存放在存储器中的一组具有相同类型的标量数据。通常,向量元素是有序的,相邻元素之间的地址增量是固定的,称之跳距(*Stide*)。

向量处理机(*Vector processor*)是执行向量操作的硬件资源,包括向量寄存器、流水线功能部件、处理器部件和寄存器计数器。对向量进行算术或逻辑运算称为向量处理。一般向量处理

比标量处理更快更有效。流水线处理机和SIMD计算机都可以执行向量操作。向量处理减少了管理循环控制的软件开销和存储器访问冲突，它利用流水线和段的概念，每个时钟周期能连续输出一个结果。向量处理机的性能提高，显然是以增加硬件和编译器的成本为代价的。能进行向量化的编译器称为向量化编译器(*Vectorizing compiler*)或者简称向量化器(*Vectorizer*)。向量指令类型有六种：

① 向量-向量指令(*Vector-Vector instruction*)。它从有关的向量寄存器取出一个或两个向量操作数，进入流水线功能部件，产生的结果放入另一个向量寄存器，可用下面两个映射来定义这些指令

$$f1:\quad V_i \rightarrow V_j \tag{1-9}$$

$$f2:\quad V_j \times V_k \rightarrow V_i \tag{1-10}$$

例如 $V1=\sin(V2)$ 和 $V3=V1+V2$ 分别是映射 $f1$ 和 $f2$，$V_i(i=1,2,3)$ 是向量寄存器。

② 向量-标量指令(*Vector-Scalar instruction*)对应于映射的向量-标量指令有

$$f3:\quad S \times V_i \rightarrow V_j \tag{1-11}$$

其中 S 为标量。

③ 向量-存储器指令(*Vector-memory instruction*)，对应于向量取和向量存，其定义如下：

$$f4:\quad M \rightarrow V \quad \text{向量取} \tag{1-12}$$

$$f5:\quad V \rightarrow M \quad \text{向量存} \tag{1-13}$$

④ 向量归约指令(*Vector reduction instruction*)，对应于如下映射

$$f6:\quad V_i \rightarrow S_j \tag{1-14}$$

$$f7:\quad V_i \times V_j \rightarrow S_k \tag{1-15}$$

从一个向量的所有元素中找出最大值、最小值和中间值是 $f6$ 的例子。$f7$ 的一个典型例子是点积，即 $S=\sum_{i=1}^{n} a_i \times b_i$

⑤ 收集和散播指令(*gather and scatter instruction*)。这些指令用两个寄存器收集随机地分布在整个存储器的向量元素或把向量元素随机地散播到整个存储器，它对应于以下的映射

$$f8:\quad M \rightarrow V1 \times V0 \quad \text{收集} \tag{1-16}$$

$$f9:\quad V1 \times V0 \rightarrow M \quad \text{散播} \tag{1-17}$$

收集操作根据变址值把存储器中的某个稀疏向量的非零元素取出放到向量寄存器中。散播则执行相反的操作，它把一个向量以稀疏向量形式存入存储器，其非零项由变址值指出。向量寄存器 $V1$ 存放数据，向量寄存器 $V0$ 存放变址值，以便获得收集或散播数据的存储器地址。收集和散播分别如图 1-11(a)、(b)所示。地址寄存器 A_0 存放存储器的基地址，向量长度寄存器 V_L 的内容表示要传送的元素个数。

⑥ 屏蔽指令(*masking instruction*)这类指令利用屏蔽向量(*mask Vector*)把一个向量压缩或展开成为一个较短或较长的索引向量，其对应如下映射：

$$f10:\ V_0 \times V_m \rightarrow V_1 \tag{1-18}$$

其中屏蔽寄存器 V_m 存放测试寄存器 V_0 的内容是零元素还是非零元素的结果，V_L 指明被测试向量的长度。V_1 寄存器存放非零元素的变址值。

收集、散播和屏蔽指令在处理稀疏矩阵或稀疏向量时非常有用。稀疏矩阵是大多数元素为0的矩阵。先进的向量处理机用硬件直接实现这些指令。