**第二章 指令系统**

**（一）指令集结构概述**

**指令：**操作码、寻址方式、操作数、寻址方式、操作数

**指令集：**

1. 一些指令的集合
2. 每条指令都是直接由CPU硬件执行

**指令的表示方法：**

二进制格式。

物理存储空间组织方式是位、字节、字和多字等

当前指令字长有16、32、64位

可变长格式和固定长度格式。（绝大部分RISC处理器其指令的长度都是确定的：MIPS32位；而以X86为代表的CISC处理器的指令长度不固定：最短1个字节，最长17个字节）。

**指令的特点：**

1. 指令的操作十分简单，其操作由操作码编码表示。
2. 每个操作需要的操作数个数为0-3个不等

操作数是一些存储单元的地址（典型的存储单元有：主存、寄存器、堆栈和累加器）

1. 操作数地址隐含表示或显示表示。（有的指令的操作数隐含在指令的操作码里面，操作码指明的功能说明了其需要处理的数据，e.g X86的PUSHX：把特殊寄存器flag的内容压到堆栈里面，无操作数）

**指令集与计算机的性能**（P38下CPU时间公式）

指令集抽象和进一步描述——>各种程序设计语言(源程序)—编译器的生成—>目标代码（指令集的二进制表示）—优化编译器（提高目标代码的质量）—>优化过的目标代码——>指令译码（提取信息）——>（为了提高指令的执行和存储效率，研究指令的编码策略）——>执行指令时，指令集的好坏影响CPI和数据通路复杂度

**（二）指令集结构的分类**

**一般从如下五个因素考虑对计算机指令集结构进行分类，即**：

1. **在CPU中操作数的存储方法（主要分类依据）**
2. 指令中显示表示的操作数个数
3. 操作数的寻址方式
4. 指令集所提供的操作类型
5. 操作数的类型和大小

**CPU中用来存储操作数的存储单元主要有**：堆栈、累加器（现在用的不多）、一组寄存器

指令中的操作数可被明确显示给出，也可按照某种约定隐式给出

（例子P367/表A-1）

**早期大多数采用堆栈或累加型指令集结构，1980年以来大多数采用寄存器型（虽然数量增加但是速度快），原因：**

1. 集成电路技术飞速发展（VLSI：very large scale integration）
2. 寄存器和CPU内部其他存储单元一样，要比存储器快很多
3. 随着编译技术的成熟，已形成一套有效分配和使用寄存器的算法（无需程序员考虑）

**通用寄存器型指令集结构的主要优点：**

1. 在表达式求值方面，比其他类型指令集结构都具有更大的灵活性
2. 寄存器可以用来存放变量：
3. 减少mem的通信量，加快程序的执行速度（因为reg比mem快）
4. 可以用更少的地址位来寻址reg,从而可以有效改进程序目标代码大小（reg的数量远远小于存储单元的数量）

**两种主要的指令特性能够将通用reg型指令集结构（GPR）进一步细分（表A-2）：**

1. ALU指令有两个或是三个操作数
2. 一条指令里最多允许有多少存储器操作数

可将当前大多数通用reg型指令集结构进一步细分为三种类型：R-R、R-M、M-M（无）

**R-R型（0,3）：优点比缺点更加明显——流水线技术多条指令可重叠执行，所以指令条数多可以掩盖掉 MIPS、PowerPC——RISC**

**优点：**指令字长固定，指令结构简洁，各种指令执行时钟周期数相近（数据通路相对简单）

**缺点：**与指令中含mem操作数的ISA（指令系统结构）相比，指令条数多，目标代码不够紧凑，因而程序占用空间较大（LOAD指令和STORE指令）

**R-M型（1,2）：X86——CISC**

**优点：**可在ALU指令中直接对mem操作数引用，无需load加载，易编码，目标代码紧凑

**缺点：**

1. **因为有一个操作数用来保存结果，所以指令中两个操作数不对称。**
2. **在一条指令中同时对reg操作数和mem操作数编码，可能限制指令所能够表示的reg个数（因为mem地址的字段比较长，而指令长度有限，所以分配给reg的字段有限）**
3. **指令的执行时钟周期因操作数的来源（reg或mem）的不同差别较大，影响数据通路和流水线**

**M-M型 (3,3)：**

优点：目标代码最紧凑，不需要设置存储器保存变量

缺点：

1. 指令字长变换很大，特别是3个操作数指令，寻址方式复杂时，字长会非常长。
2. 每条指令完成的工作差别也很大
3. 对Mem频繁访问使mem访问成为性能瓶颈——这种类型已不用了（现在CPU和mem速度差异越来越大）

**指令集结构设计概观：硬件——>指令集——>操作系统软件——>应用程序软件（步骤）**

**（三）寻址技术**

在通用reg型指令集结构中，一般利用**寻址方式**指明指令中的操作数（存放在哪里）是一个**常数**、一个**寄存器操作数**或是一个**存储器操作数。**

寻址是从形式地址（由指令描述）到实际地址（有效地址——寄存器地址或mem单元的地址）转换。——必须加速有效地址的生成**（表A-4）**

最常用：立即数和位移量寻址；ALU运算和比较指令易采用立即数寻址。

**（四）指令系统的设计和优化**

一种指令集结构中的指令到底要支持哪些类型的操作？——指令集结构功能设计问题（操作码）

**指令集操作的分类（书P376/表A-5）**

1. **控制指令：**

**跳转（Jump）：**当控制指令为无条件改变控制流时，称之为“跳转”。

**分支（Branch）：**当控制指令是有条件改变控制流时，称之为“分支”

**控制流程的改变情况：**条件分支（频率最高）、跳转（最低）、过程调用、过程返回（其次）

**条件分支指令的表示**（书P379/表A-7）

**分支目标地址的表示**（偏移量较小<8位——短转移，转移范围不大）

**过程调用和返回的状态保存：（补充见本）**

**“调用者保存”方法：**如果采用调用者保存策略，那么在一个调用者调用别的过程时，必须保存**调用者所要保存的reg（子程序执行过程中可能会破坏这些状态）**，以备调用结束返回后，能够再次访问调用者。

**“被调用者保存”方法：**如果采用被调用者保存策略，那么被调用的过程必须保存它要用的寄存器，保证不会破坏过程调用者的程序执行环境，并在过程调用结束返回时，恢复这些寄存器的内容。

**（五）指令系统的发展和改进**

1. 一个方向是**强化指令功能**，实现软件功能向硬件功能转移，基于这种指令集结构而设计实现的计算机系统称为**复杂指令集计算机（CISC）**.——X86中很多功能性强指令，如串操作指令等

2. 八十年代发展起来的**精简指令集计算机（RISC），**其目的是尽可能地降低指令集结构的复杂性，以达到简化实现，提高性能的目的。

**CISC指令集功能设计：**

1. **面向目标程序增强指令功能**
2. 提高运算型指令功能（嵌入式（乘累加很普遍）处理器的MAC指令——可计算D=A\*B+C）
3. 提高传送指令功能
4. 增加程序控制指令功能
5. **面向高级语言和编译程序改进指令系统**
6. 增加对高级语言和编译系统支持的指令功能
7. 高级语言计算机指令系统
8. **面向操作系统的优化实现改进指令系统（主要思想：将耗时的软件操作用硬件实现）**
9. 主要表现在对操作系统耗时的操作如中断处理、进程管理、存储管理和保护、系统工作状态的建立与切换等的支持。**（随着虚拟化技术的流行，Intel和AMD都推出了硬件虚拟化技术，该技术通过在指令系统中增加专门的虚拟化指令实现，虚拟化管理器中耗时的操作可用指令实现）**
10. 可以设置支持系统工作状态和访问方式转移的指令、支持进程转移的指令、支持进程同步和互斥的指令等措施，达到优化实现操作系统的目的

**目前朝着RISC方向发展，原因：**

1. **CISC结构存在如下缺点：**
2. 各种指令的使用频率相差悬殊（20%频率很大，占运行时间80%）
3. CISC结构指令系统的复杂性带来了计算机体系结构的复杂性，这不仅增加了研制时间和成本，而且还容易造成设计错误。

**RISC指令集功能设计：（奔腾开始，X86开始实现了指令RISC化）**

1. CISC结构指令系统的复杂性给VLSI设计增加了很大负担，不利于单片集成。
2. CISC结构的指令系统中，许多复杂指令需要很复杂的操作，因而运行速度慢。
3. 由于各条指令的功能不均衡性，不利于采用先进的计算机体系结构技术（如流水技术）来提高系统的性能。

目标：使计算机体系结构更加简单、合理和有效

**RISC指令集功能设计原则：**

1. 选取使用频率最高的指令，并补充一些最有用的指令
2. 每条指令功能尽可能简单，并在一个机器周期内完成
3. 所有指令长度均相同
4. 只有load和store操作指令才访问mem,其他指令操作均在regs之间进行;

**（六）操作数的类型和大小**

**1.** 操作数类型和操作数表示是**软硬件主要界面之一**。

2**.操作数类型**是面向应用、软件系统所处理的各种数据结构

3**.操作数表示**是硬件结构能够识别、指令系统可以直接调用的那些结构

4.**操作数表示所表征的那些操作数类型，**是应用软件和系统软件所处理的**操作数类型**的子集。

5.确定操作数表示实际上也是软硬件取舍折中的问题：

1）计算机即使只有最简单的操作数表示（如只有整数（定点）表示法），也可以通过软件方法处理各种复杂的操作数类型，但这样会大大降低系统效率

2） 如果各种复杂操作数类型均包含在操作数表示之中，虽然会大大提高系统效率，但所化给的硬件代价也很高。

**整数（定点）：**二进制补码表示；大小可以是字节、半字或单字（32位）

**浮点：**阶码和尾数；可分为单精度浮点（单字大小）和双精度浮点（双字大小）；当前普遍采用的是IEEE 754 浮点操作数表示标准

**字符和字符串：**8位ASCII码表示

**十进制：**“压缩十进制”或“二进制编码十进制”，压缩和解压。

**操作数类型的表示主要有以下两种方法：**

1. 操作数的类型可以由操作码的编码指定——最常见
2. 数据可以附上由硬件解释的标记，有这些标记指定操作数的类型，从而选择适当的运算——有标记数据的机器很少见

一般操作数类型大小选择有：字节、半字、单字（整数最高）和双字（浮点数最高）

**（六）MIPS指令系统结构**

1. Load/Store型指令集结构

2.是一种多元指令集结构，还将会体现未来一些机器的指令集结构的特点

**特点：**1. 具有一个简单的Load/Store指令集

2．注重指令流水效率

3.简化指令的译码

4.高效的支持编译器

1.**寄存器：**

32个32位的通用寄存器（GPRs）寄存器R0的内容恒为全0

32个32位单精度浮点寄存器（FPRs）——16个64位双精度浮点寄存器

1. **数据类型：**

**整型数据：8位、16位、32位**

**浮点数据：32位单精度、64位双精度、IEEE754标准**

1. **寻址方式—**寄存器寻址、立即数寻址、偏移寻址、寄存器间接寻址（mem地址宽度32bits）
2. **指令格式三种（书P390）**
3. **MIPS操作类型：**Load和Store操作、ALU操作、分支和跳转操作、浮点操作

**“<——n”：**传送n位数据**；“##”：**两个域的串联操作**；域下标和上标（见本）变量Mem**表示存储器中的一个数组，存储器按照字节寻址，可以传送任何数目的字节

**Load和Store操作：对R0无效（P391/表A-9）（浮点数操作大量使用浮点载入，整型加主要用来计算访存地址）**

**ALU操作：**都是reg-reg型指令，包含简单算逻运算。

“设置相等”“设置不等”“设置小于”：reg比较指令，比较为真，目标寄存器填1

**描述目标地址的方法：**

1. 其中两种类型的跳转指令用**带符号位的26位偏移量加上PC的值**来确定跳转的目标地址。
2. 另外两种类型则**指定一个寄存器（作为J类型的操作数）**，由寄存器中的内容决定跳转的目标地址

**两种跳转类型：**

1. 简单跳转
2. 跳转并链接（用于过程调用）：除了完成转移之外，将下一条顺序指令地址（返回地址）保存在R31中（J R31）——代替X86的call

浮点操作：操作数来源于浮点寄存器，同时指明操作是单精度还是双精度（D代表双精度浮点操作，后缀F代表单精度浮点操作）

**MIPS的效能分析：**MIPS（RISC）和VAX（CISC）作对比，大概是其1.5倍

**影响计算机性能的因素有三个：**虽然**程序指令数**增加了，但是**执行每条指令所用的时间缩短**了（因为MIPS采用了流水线技术，而且**处理器的主频**大大提高了）

**第三章 流水线技术**

**3.1流水线基本概念**

**1.计算机中的流水线**

指令流水线：入——分析阶段（分析器）——执行阶段（执行部件）——出

功能部件流水线：功能部件按照流水方式组织和运行

**2流水技术**

上世纪六十年代计算机硬件资源很昂贵，通过流水技术减少硬件资源是降低成本的很好方式

将一重复的时序过程**分解成若干子过程（分解方法根据具体过程）**，每个子过程都可有效地在其专用功能段上与其它子过程**同时执行**，这种技术成为流水技术。

**3.流水线的特点**

（1）流水过程由多个相关的子过程组成，这些子过程称为流水线的“**级**”或“**段**”。段的数目称为流水线的“**深度**”。（深流水处理器）

（2）每个子过程由专用的功能段实现。

（3）各功能段的时间应基本相等，通常为1个时钟周期（1拍）

（4）流水线需要经过一定的**通过时间**（从开始到满载）才能稳定。

（5）流水技术适合于**大量重复**的时序过程。（才能更体现降低资源消耗，提高吞吐率的优点）

**3.2流水线的分类**

**1.单功能流水线和多功能流水线（按流水线所完成的功能分类）**

单功能流水线：只能完成一种固定功能的流水线（e.g 功能单元流水线）

多功能流水线：指流水线的各段可以进行不同的连接，从而完成不同的功能。

**2静态流水线和动态流水线（按同一时间内流水段的连接方式）**

**静态流水线：**在同一时间内，流水线的各段只能按同一种功能的连接方式工作。E.g TIASC流水线—适合于处理一串相同的运算操作

**动态流水线：**在同一时间内，当某些段正在实现某种运算时，另一些段却在实现另一种运算。（使流水线的控制变得复杂，一些数据通路会有冲突）

**3.部件级、处理机级及处理机间流水线（按流水的级别划分）**

**部件级流水线：**又叫**运算操作流水线**，把处理机的算数逻辑部件分段、使得各种数据类型的操作能够进行流水。(部件是乘/加，部件范围内进一步流水)

**处理机级流水线**，又叫**指令流水线**，是把解释指令的过程（典型五个段）按照流水方式处理。

**处理机间流水线**，又叫**宏流水线**，是由两个以上的处理机串行地对同一数据流进行处理，每个处理机完成一项任务。

**4.标量流水处理机和向量流水处理机（按数据表示来分类）**

**标量流水处理机(大部分，RISC，MIPS)：**指数据不具有向量数据表示，仅对标量数据刘水处理（IBM360/91，Amdahl 470V/6等）

**向量流水处理机（典型的巨型机上）：**指处理机具有向量数据表示，并通过向量指令对向量的各元素进行处理。（TIASC, CRAY-1,YH-1）（近年来再次兴起）

**4.线性流水线和非线性流水线（按是否有反馈回路来进行分类）**

**线性流水线：**指流水线的各段串行连接，没有反馈回路。

**非线性流水线：**指流水线中除有串行连接的通路外，还有反馈回路。—存在流水线调度问题

**流水线调度问题：**什么时候向流水线引进新的输入，从而使新输入的数据和先前操作的反馈数据在流水线中不产生冲突，此即所谓流水线调度问题。——所以一般不采用非线性流水

**3.3 MIPS基本流水线**

**3.3.1MIPS的简单实现（P478图）**

**1.取指令周期（IF）**

（1）根据PC值从mem中取指，将指令送入IR（指令寄存器）

（2）PC值加4，指向顺序的下一条指令

（3）将下一条指令的地址（PC+4）放入临时寄存器NPC中（这里NPC是临时的是因为后续PC值可能会变化，比如遇到转移或者控制指令）

**2.指令译码/读reg周期（ID）**

（1）读IR（指令寄存器），进行指令**译码**

（2）按照相应reg号**读reg文件，**将读出结果放入两个**临时寄存器A和B**中

（3）同时对IR中内容低16位**符号扩展**，将所得的32位立即值保存在**临时寄存器Imm**中

**3.执行/有效地址计算周期（EX）**

mem访问、reg-reg ALU、reg-imm ALU、分支操作

**执行和有效地址计算可以合并的原因：**这是MIPS的设计，某条指令要么是执行某些功能的操作；要么是对有效地址进行计算，进行存储器的访存操作；要么是条件判断，新的转向地址计算；一条指令不可能对这些操作同时展开；因此只能选其一，可以合并。

1. **mem访问周期（MEM）：**访存操作、分支操作

**5.写回周期（WB）:**reg-regALU、reg-Imm ALU、load操作

**6性能分析：访存最费时间，比译码时间长的多**

在该数据通路上，分支指令需要4个时钟周期（1-4），其他指令需要5个时钟周期（1-5），CPI=4.88计算见本——不是一种优化实现！

1. **改进方法**
2. 在Mem周期完成ALU指令——假设ALU指令数占44%，在时钟周期时间不变的同时，CPI可以降低至4.4
3. 如果要进一步降低CPI，可能会增加时钟周期时间（因为如果降低CPI，那么每一拍要完成更多的功能，所以会增加时钟周期时间）
4. 采用单周期实现，可以将CPI降低为1，但是时钟周期却变成了原来的五倍（CPU时间依然没变）

**所以一般不采用该方法——原因：将所有工作压到一拍，资源吞吐率效率都很低；因此采用流水技术**

**3.3.2基本的MIPS流水线**

**1.一种简单的MIPS流水线（时空图见书P458,流水线可以看成是按时间错开的数据通路序列P459）：**将上面的数据通路流水化 ，使得

（1）数据通路中的每一个周期成为流水线的一段

（2）每个时钟周期启动一条指令

**2.实现流水技术应解决的一些问题：**

**（1）应保证流水线各段不会在同一时钟周期内使用相同的寄存器通路资源**

1）不能要求ALU既做有效地址计算，又做减法操作

2）IF与MEM都要**访存**，怎样避免访存冲突？——设置各自相应存储器：指令存储器和数据存储器

3）ID与WB都要**访问寄存器**，是否存在冲突，怎样避免？——资源隔离

**（2）PC计算问题：**

为了能够在每个时钟周期启动一条新的指令，流水线必须在**IF段**获得下一条指令地址，并将其保存在PC中；但是分支指令会改变PC值，且只有在**MEM段**结束时，这个新值才会被写入PC——矛盾

**解决方法：改变数据通路，在IF段完成PC计算。但分支指令如何处理？——后面讲！**

**（3）合理划分流水段，每段内的操作必须在一个是时钟周期内完成**

**（4）流水线寄存器设计（书P479/图）：**

为防止寄存器中的值在为流水线中某条指令所用时被流水线中其他指令所重写，在流水线各段之间设置**流水线寄存器文件**，也称**锁存器——把各段分开来，使得跨段间不会产生数据重写。**

**流水线寄存器作用：**当指令在流水线中流动时，其数据和控制信息也在同步地向前流动。同时不影响左右两段的执行，达到封闭满足本段执行的功能。

**3.MIPS流水线的操作：**

任一时刻，流水中的指令仅在流水线中的某段内执行操作。因此，只要知道每一流水段在各指令下进行何种操作，就知道了整个流水线的操作。（表C-9 P480）

**4.MIPS流水线的多路选择器：**

ALU输入端俩个MUX——由ID/EX.IR指出的指令类型控制

IF段的MUX——由EX/MEM.Cond域的值控制

WB段的MUX由当前指令类型（Load/ALU）控制

* + 1. **流水线性能分析**

**三项性能指标：吞吐率、加速比和效率**

1. **吞吐率——衡量流水线速度的重要指标：**指单位时间内流水线所完成任务数或输出结果的数量。

**最大吞吐率TPmax：**指流水线在达到稳定状态后得到的吞吐率（每段都忙得时候）；取决于**最慢**一段所需的时间，该段是**流水线的瓶颈**

**消除瓶颈的方法：**（1）细分瓶颈段（2）重复设置瓶颈段（同一位置设置两套流水段的硬件资源）——不希望

设流水线由m段组成，完成n个任务的吞吐率称为**实际吞吐率TP**

1. **加速比：**流水线速度与等功能的非流水线速度之比。
2. **效率：流水线的设备利用率**

由于流水线有通过时间（起步时间）和排空时间，所以流水线各段不是一直满负荷工作，E<1

**静态流水线实现向量乘：**

（1）静态多功能流水线在对某种功能进行处理时，总有些段处于空闲状态

（2）功能切换增加了前一种功能的排空时间和后一种功能的通过时间

（3）需要把输出回传到输入（相关——先不考虑）

流水线额外开销包括：流水寄存器的延迟（建立时间和传输延迟）以及时钟扭曲。

**有关流水线性能的若干问题：**

1. 流水线不能减少（而且一般是增加）单条指令的执行时间，但能够提高吞吐率（并行执行指令条数）。
2. 增加流水线的深度可以提高流水线性能（切分段数量多，**每个段时间短**，效率加速比高）
3. 流水线深度受限于流水线的延迟和额外开销（切分的越多增加的开销越多）
4. 需要用高速锁存器作为流水线寄存器
5. 指令之间存在的相关，限制了流水线的性能
   * 1. **流水线中的相关**

**1.流水线中的相关**是指相邻或相近的两条指令因存在某种关联，后一条指令不能再原先指定的时钟周期开始执行。——基本方法：暂停：暂停流水线中某条指令及其后面所以指令的执行，该指令之前的所有指令继续执行。

**2.三种不同类型的而相关**

**结构相关：**当指令在重叠执行过程中，**硬件资源**满足不了指令重叠执行的要求，发送资源冲突时将产生“结构相关”。

**数据相关：**因一条指令需要用到前面指令的结果，而无法与产生结果的指令重叠执行时，发生数据相关。**（数据使用时间）**

**控制相关：**当流水线遇到分支指令和其他会改变PC值的指令时可能发生控制相关。

**（一）流水线的结构相关**

1. **导致结构相关的常见原因：**
2. 功能部件不是全流水（能够并行容纳的指令有限，不能像流水那样每段都容纳那么多）
3. 重复设置的资源不足（并行执行指令的方式——瓶颈段）
4. 当数据和指令存在同一存储器中时，访存指令会引起**存储器访问冲突。（书P463）**

**2.避免结构相关的方法：**

1. 所有功能单元完全流水化。（每一段都可以容纳一条指令，因此这个功能部件可以容纳多条指令执行，所以资源冲突可以得到很大缓解）。—把资源批成很多小的段
2. 设置足够多的硬件资源——但是，硬件代价很大！

**3.有些设计方案允许结构相关存在——**降低成本、减少功能单元的延迟（因为功能单元不需要重复设置）

**（二）流水线的数据相关（每个时钟周期前半周期写入，后半周期读取）**

**1.产生原因**：当指令在流水线中重叠执行时，流水线有可能改变指令读/写操作数的顺序，使之不同于他们在非流水实现时的顺序，这将导致数据相关。——最简单：插入暂停周期

2.通过**定向技术**（forwarding，旁路bypassing）减少数据相关带来的暂停

**主要思路：**将计算结果从其产生的地方直接送到真正需要它的地方，就可以避免暂停。

1. 寄存器文件EX/MEM中的ALU运算结果总是送到ALU的输入寄存器
2. 从定向通路得到输入数据的ALU操作不必从源寄存器中读取操作数

**进一步推广：**一个结果不仅可以从某一功能单元的输出定向到其自身的输入（ALU），而且可以定向到其他功能单元的输入。

MIPS中，任何流水寄存器—>任何功能单元的输入都可能需要定向路径，将形成复杂的旁路网络。

1. 两条指令**访问同一存储单元**，也可能引起数据相关，例如访问数据Cache失效时。（不涉及！）——本章只讨论**寄存器数据相关！**
2. **数据相关的分类**

（1）RAW——最常见的数据相关，严重制约CPU性能，是程序最重要特征之一。

（2）WAR——**MIPS流水线不会出现这种相关（有硬件控制）；当有些指令在流水段的前半部分写结果，后半部分读源操作数，可能引起这种类型相关。**

**（3）WAW——MIPS流水线不会出现这种相关（有硬件控制）；当流水线中有多个段可以写回，而且当流水线暂停某条指令的执行时，其后的指令可以继续前进时，可能引起这种类型的相关。**

1. **需要暂停的数据相关——**并非所有的数据相关都可以通过定向技术解决。（书P468例子）——解决：增加流水线“**流水线互锁**”部件，当互锁硬件发现这种相关后，暂停流水线，直到相关消除。暂停时钟周期叫做“**载入延迟**”（P468/下表C-3）
2. **对数据相关的编译调度方法：**流水线中常常会遇到多种类型的暂停；编译器可以通过重新排列代码的顺序来消除这种暂停，为“**流水线调度**”或“**指令调度**”。
3. **对MIPS流水线控制的实现：**

**（１）指令发射：**指令从流水线的译码段进入执行段的过程

**（２）检测数据相关**：

1）ID段可以检测所以数据相关

2）在使用一个操作的时钟周期开始（EX和MEM）检测相关，并确定必须定向

LOAD互锁：在ID段检测是否需要启动Load互锁，必须进行三种比较

一旦检测到相关，控制部件必须在流水线中插入暂停周期，使IF和ID段中的指令停止前进——**ID/EX中控制部分清0、保存IF/ID内容不变**

**所有的定向都是从ALU/DM的输出到ALU、DM或0检测单元的输入**

**第四章 指令级并行**

**1.指令级并行（ILP）：**当指令之间不存在相关时，他们在流水线中是可以重叠起来并行执行的。这种指令序列中存在的潜在并行性称为~

软硬件技术都可以提高ILP，二者相互配合，才能最大限度挖掘出程序中存在的ILP

**2.本章研究的技术(主要是硬件支持)及克服的停顿：**

基本流水线调度——RAW

循环展开——控制相关

寄存器换名——WAW和WAR

指令动态调度（记分牌和Tomasulo算法——还有控制相关）——各种数据相关

动态分支预测——控制相关

推测——所有数据/控制相关

多指令流出（超标量和超长指令字）提高理想CPI（<1）

**所有技术必须和软件，特别是编译器合作完成。**

**3.几个基本概念**

**基本（程序）块：**一段除了入口和出口以外不包含其他分支的线性代码段

1. 每6-7条指令有一个分支
2. 必须在多个基本块之间开发ILP

循环级并行：循环体中指令之间的并行性

开发循环级并行的基本技术方法：指令调度、循环展开、寄存器换名

**4.1循环展开调度的基本方法**

1**.循环展开**是展开循环体若干次，将**循环级并行转化为指令级并行**的技术

2.这个过程即可以通过编译器**静态**完成，也可以通过硬件**动态**进行

3.本章中的分支指令为条件转移指令

4**.编译器在完成这种指令调度时，受限于以下两个特性：**

1）程序固有的ILP

2）流水线功能部件的执行延迟（约定见116页表3-2）

**5.流水线其他特性说明：**

1）整数流水线采用改进的MIPS整数流水线（取存的相关通过旁路机制解决）

2）分支指令，由整数流水线执行；

（1）分支条件的检测调整到ID段（以往有一些是在EX段）

（2）如果分支指令使用**上一条指令**的结果作为分支条件，将要延迟1节拍

（3）分支指令有1个节拍的**延迟槽**

3）浮点运算一般为64位

1. 无结构冒险

**6.P116例子编译过程**：每一遍循环之间是不存在相关的，多遍循环可以同时执行而不会导致结果的错误

**7.循环展开总结：**

（1）对指令移动有效——展开有用

（2）用不同寄存器（换名/更多寄存器）

（3）消除额外测试开销

（4）在循环展开时要分析Load/Store指令——访存，需要动态识别不能静态识别，地址存在不确定性（内存地址换名）

1. 保留真相关（RAW）

**4.2指令的动态调度**

1. 编译器本质上通过对每个循环迭代中**寄存器重命名**来展开循环
2. 硬件也可通过**寄存器重命名和乱序执行**来获得同样的效果
3. 动态调度：记分牌、Tomasulo’s算法

**4.冒险的检测和调度：**

（1）如果存在数据相关，硬件检测机制会做如下的事情直到相关可能带来的错误

1）暂停指令

2）停止取指令和发射指令

（2）静态调度和动态调度混合工作：

1）软件负责调度指令减少空转（静态调度）

2）硬件对静态难以识别的ILP的执行重新排序来减少空转（动态调度）

5.**动态调度的目的：**在**程序执行**的时候，解决WAW，WAR和RAW带来的冒险

**优点：**（1）硬件执行时处理在编译的时候未知的相关，简化编译器

（2）在不同的流水线上都能有效的运行

缺点：大大增加硬件复杂性

1. **记分牌技术**
2. **记分牌1964被Cray用于CDC 6600机器**
3. **记分牌允许指令乱序执行，前提**
4. 充足的资源，无数据相关
5. 记分牌动态解决了RAW相关
6. 指令可以乱序执行
7. **基本原理：**
8. 每条指令均经过记分牌，记录各指令间数据相关的信息
9. 如果记分牌判断出一条指令不能立即执行，他就检测硬件的变化从而决定何时执行
10. **记分牌处理：**

**ID段分为两级：**

1. **流出**—解析指令，检查结构相关
2. **读操作数**一直到不存在数据相关时，才读取操作数

如果存在WAR或WAW相关，记分牌暂停这条指令的执行，直到相关消除后才继续执行

1. **记分牌执行过程**

**流出（满足不了以下两个条件将会阻塞在指令队列中，否则流出到功能部件中）：**

1. 本指令所需的功能部件有空闲
2. 正在执行指令使用的目的寄存器与本指令的目的寄存器不同（保证**WAW**相关）

**读操作数（需满足以下两个条件之一，否则阻塞在功能部件中）：——动态解决RAW相关，任何一条指令的源操作数产生之前，这条指令都不能被执行**

1. 前面已流出的还在运行的指令不对本指令的源操作数寄存器进行写操作**（RAW）**
2. 一个正在工作的功能部件已经完成了对这个寄存器的写操作

**这两步完成原来ID段的功能**

**执行：**

1. 开始于取到操作数以后
2. 当结果产生后，修改记分牌状态
3. FP流水部件会占用多个周期（一拍出不来结果）

**写结果：检查WAR相关**

**出现以下情况时，不允许指令写结果：**

1. 前面的某条指令还没有读取操作数
2. 其中某个源操作数寄存器与本指令的目的寄存器相同
3. **Tomasulo算法**

**（一）产生背景**

1. **IBM360/91**比CDC6600晚三年推出，商业计算机使用Cache之前最大型的一台机器
2. **IBM要求整个360系列仅一个指令系统和一个编译器——保证体系结构不变**
3. 要求具有很高的浮点性能，但不是通过高端机器的专用编译器实现（对体系结构的设计要求苛刻）
4. 只有四个双精度浮点寄存器，编译器调度的有效性受到很大限制
5. 没有Cache，机器的访存时间和浮点计算时间都很长
6. 可支持循环的多次迭代重叠执行（在体系结构不变的情况下）

**（二）Tomasulo算法与记分牌**

**1.采用了许多记分牌中的理念**

**2.两个较大的差异：**

（1）Tomasulo算法，冲突检测和执行控制是分布的，利用保留站实现（原通过记分牌完成）

（2）Tomasulo算法不检查WAW和WAW相关，他们通过算法本身消除掉了—很好解决名相关

**（三）MIPS上实现算法**

**1.360/91浮点功能单元：**3个加法单元、2个乘法单元、6个读单元、3个写单元（MIPS上也是）

**2.MIPS与360/91浮点单元的区别：**

（1）360/91:支持R-M指令（CISC），MIPS只支持R-R指令

（2）360/91:采用流水化功能单元，不采用多个功能单元（MIPS）

**3.每个功能单元都要保留站：缓冲**

**4.MIPS五阶段的流水线的改造**

**（1）ID和EX段被以下三个阶段代替**

1.流出 2.执行 3.结果写回

**流出：**

1. 从浮点指令队列中取出一条指令
2. 根据这条指令译码产生的结果，找到其所需要的FU；看该FU是否有空保留站，若有则流出这条指令
3. 如果操作数在寄存器中，就送到该指令对应的保留站中；否则等待
4. 存储器存/取指令只要有空闲缓存就可以流出
5. 如果没有空闲的保留站或者缓存，就存在结构相关，指令暂停，直到有空闲的保留站或者缓存

**执行：**

1. 如果缺少一个或者多个操作数，执行工作就不能执行，监听CBD

——**这个阶段实际就是检测和自动维护RAW相关**

1. 如果两个操作数都就绪，这条指令就可以执行

**结果写回：**

1. 如果结果已经产生，将其写到CBD上
2. 通过CDB，把这个结果写到目标寄存器和等待这个结果的所有功能单元的保留站

**（四）Tomasulo算法的优点**

1.分布式硬件冲突检测——相关冲突等检测

2.利用保留站和缓冲完成寄存器换名**，彻底消除WAW和WAR两种名相关**

3.如果多个保留站等待同一个操作数，当操作数在CDB上广播时，他们可以同时获得所需要的数据

**动态调度方法评价：**

**优点：**能够达到很高的性能

**缺点：（1）高复杂性：需要大量硬件**

**（2）存在瓶颈：**单个CDB引发竞争（乱序结束，更多FU同时完成操作）——现在额外增加CDB（3条，两条局部（浮点一条整数一条，整浮之间一条）一条全服）——缺点在每个保留站上需要为每条CDB设置重复的硬件接口

**4.3控制相关的动态解决技术——硬件**

指令流出的时候控制相关是瓶颈，特别是流出多条指令，一次控制相关的开销巨大

**4.3.1分支预测缓冲**

**（一）动态分支预测**

1.动态分支预测的两个理由：

（1）n流出的处理器加速上限为n倍

（2）Amdahl定律显示，在较低CPI机器上，控制相关导致的空转对机器性能影响大

**2.前面解决控制相关的静态策略（指令调度）：**

需要编译器将一条或多条指令移动到流水线产生的分支延迟槽中

**3．关于分支预测策略的两部分工作：**

（1）预测的分支是否成功

（2）执行分支目标指令（分支目标指令在什么地方）

**4.分支预测的效率**

（1）预测的准确率

（2）分支的开销：预测正确的开销/预测错误的开销

**5.分支预测缓冲（BPB）：原理**

（1）最简单的分值预测策略**（上次成功这次也成功）**

**（2）分支预测缓冲是一个小的存储器阵列**

1）每个存储单元单元只有1位，记录最近一次分支是否成功的信息

2）预测位为1则预测分支成功，并从目标位置开始取指令

3）**单元由分支指令地址的低位索引进行寻址**

4）BPB的预测位会被具有相同低位地址的分支设置，两个指令之间会产生冲突

5）未被选中时，得到两次错误预测，因为错误预测导致预测位反转

（3）BPB也被称为BHB(分支历史缓冲)

**6.2位BPB工作原理——一个预测必须错误两次才会改变**

**7.BPB实现**

**（1）BPB实现方案**

1）实现一个小而特殊的“Cache”,利用指令地址进行索引，在IF流水段访问

2）或者，为指令cache中每一块增加附加位，与指令一起取出

**（2）若一个指令在ID段被译码为分支指令，且对应BPB标志位预测成功，则**

1）一旦PC已知，立刻从分支目标地址开始取指

2——或者，继续顺序取指

**4.3.2分支目标缓冲（见本）**

**BTB和BPB的比较：**

1. BPB受限于预测精度，以及预测失效后产生的开销
2. BPB预测精度80%,BTB更高80%-95%
3. 较低失效开销技术：在一个时钟周期内将分支预测正确和错误的两条指令都取出来（不同分支路径）

（1）缺点：这种做法虽然增加了机器执行的速度，但是也会引入其他开销，如存储系统的硬件端口加倍

（2）降低失效开销的唯一方法（如AS/400 PowerPC处理器上）：多分支预测技术

**分支预测局限性：**

1. 预测准确性：80%-90%
2. 预测性能依赖于程序类型和缓冲区大小
3. 预测失效开销的优化：预取不同分支路径指令（存储端口数目加倍，交叉存取缓冲）、在目标缓冲中缓冲多个路径的指令

**4.4多指令流出技术**

多指令流出处理器——实现一个时钟周期内流出多条指令时，达到CPI<1

1. **多流出处理器2种基本结构**

**1.超标量：**每个时钟个周期流出指令数不定；可以编译器静态调度也可以硬件动态调度实现

**2.超长指令字（VLIW）:**每个时钟周期流出的指令数是固定的，只能通过编译静态调度

**（二）超标量处理机**

**1.原型来自IBM的“America”处理器：**RS/6000第一个采用超标量技术，现在几乎所有高性能处理器都采用该技术

**2.超标量处理机的硬件支持每个时钟周期发出1-8条不存在相关的指令**，如果指令流中的指令相关或不满足限制条件，则只能流出这条指令前面的指令，因此超标量处理器流出的指令数是不定的。

**（三）超长指令字技术**

1.一台超标量机器每周期能流出4-8条指令，由于必须要用硬件分析指令间相关，为其实现带来了困难

2.另一种选择：VLIW指令个数固定，软件编译器指定固定位置流出（AP-120B,FPS-164）

**3.VLIW基本结构：**

（1）采用多个独立功能单元，多个不同操作封装在一条长指令字中，每个功能单元在VLIW中有一定的对应区域（16-24位）

（2）VLIW硬件只是简单地将指令字中对应的部分送给各个功能单元，功能单元在哪一个时钟周期执行什么操作是由编译器决定，硬件不处理——问题：很多指令不能配套执行

**（3）VLIW的技术难题：**

1. 从线性代码片段中产生足够的操作需要进行激进的循环展开，这增大了代码大小

2.无论指令是否被充满，没有被使用的FU也在指令字编码过程中占据了相应的位。这不仅浪费运算资源，还占据了大量的存储空间——解决：在主存中压缩指令，在cache中解压缩指令

3VLIW带来了二进制代码兼容性问题（2个浮点部件和3个浮点部件）——模拟方式来解决.

4.需要使用大量的寄存器

**第七章 多处理机(MIMD)（中小规模的机器：processor数<100）**

**引言：**

**并行计算机在未来将会发挥更大的作用：**

1. 获得单处理机性能，最直接就是把多个处理器连在一起
2. （1985年以来）体系结构改进使性能迅速提高，但通过复杂度和硅技术的提高而得到的性能提高正在减小。
3. 并行计算机应用软件已有缓慢但稳定的发展。

**7.1.1并行计算机体系结构的分类（Flynn分类法）**

**1．MIMD已成为通用多处理机体系结构的选择，原因：**

1. MIMD具有灵活性（连接方式，可大可小）
2. MIMD可充分利用商品化微处理器（MIMD的组成）在性能价格比方面的优势

**2.MIMD机器分为两类（每一类代表了一种mem的结构和互联策略）：**

1. **集中式共享存储器结构**

有时被称为UMA（uniform memory access）机器。（书P259/图5-1）UMA:所有处理器访问mem的延迟都是一致的

1. **分布式共享存储器结构：**
2. 每个节点包括处理器、存储器和IO
3. 在许多情况下，分布式存储器结构优于采用集中式共享存储器结构
4. 分布式存储器结构需要**高带宽的互连（而非常规的连接以太网等）**

**分布式存储器结构的优点：**

1. 如果大多数的访问是针对本结点的局部存储器，则可降低存储器和互联网络的带宽要求（无需占用网络带宽）
2. 对局部存储器的访问延迟低（直接本地访问，速度最快）

**分布式存储器结构的主要缺点：**处理器之间的通信较为复杂（通过互连网络实现），且各处理器之间访问延迟较大（互连网络的延迟大）。

**7.1.2通信模型和存储器的结构模型**

**1.两种地址空间的组织方案：**

**（1）物理上分离的多个存储器可作为一个逻辑上共享的存储空间进行编址（**适合可变规模的机器情况，少或多时都能较好适应）

1）这类机器的结构被称为分布式共享存储器（DSM）或可缩放共享存储器体系结构。

2）DSM机器被称为NUMA（non-uniform memory access）机器（通过互联网络访问存储器，尽管是同一个地址空间，但是访问本地和访问远程所用的时间是不一样。）

**（2）整个地址空间由多个独立的地址空间构成，他们在逻辑上也是独立的，远程的处理器不能通过地址访问对其直接寻址**

每一个处理器-存储器模块实际上是一个单独的计算机，这种机器也称为多计算机。

**2.两种通信模型：**

**（1）共享地址空间的机器：**利用Load和Store指令中的地址隐含地进行数据通信。（访问对用户完全透明，在底层，访问本地访问远程用户并不清楚）

**（2）多个地址空间的机器：**

**1.**通过处理器间**显式地**传递消息完成（消息传递机器）。

2.代价高时间长：消息传递机器根据简单的**网络协议**，通过传递消息来请求某些服务或传输数据，从而完成通信。

**e.g 一个处理器要对远程mem上的数据进行访问或操作（发送、接收、应答过程）：**

（1）发送消息，请求传递数据或对数据进行操作，远程进程调用（RPC, remote process call）——（通过调用的方式把数据传送到远程数据需求的地方。）

（2）目的处理器接收到消息后，执行相应的操作或代替远程处理器进行访问，并发送一个应答消息将结果返回。

对于通信机制的性能，通过下面三个关键的性能指标进行衡量：

1. **通信带宽（最重要）——**理想状态下的通信带宽受限于处理器、存储器和互联网络的带宽。进行通信时，节点内与通信相关的资源被占用，这种占用限制了通信速度。
2. **通信延迟**——理想状态下通信延迟应尽可能地小。

通信延迟=发送开销（发送方将数据送到发送端口）+**跨越时间**（发送端口第一位数据从发送端口到接收端口传输所用的时间）+**传输延迟**（最后一位通信的数据从发送端到接收端的延迟）+接收开销（接收方将数据从接收端移到接收的进程中）

1. **通信延迟的隐藏**——将通信和计算或多次通信之间重叠起来，以实现通信延迟的隐藏

**两种通信机制各有优点，共享存储器通信主要优点：**

1. 与常用的SMP**使用的通信机制兼容**。（都是通过**指令**存取数据）
2. 当处理器通信方式复杂或程序执行动态变化时**易于编程，同时在简化编译器设计方面也占有优势**。——通信是通过指令进行，不需要过程调用
3. 有用熟悉的共享存储器模型开发程序的能力，只关注那些性能关键的访问。
4. 当通信数据较小时，通信开销较低，带宽利用较好。**(源于通信的隐含特性以及使用地址映射来实现在硬件中的保护，而非通过I/O系统——指令的通信量较少)**
5. 通过硬件控制的Cache减少了远程通信的频度，减少了通信延迟和对共享数据的访问冲突。（每个结点都有Cache）

**消息传递通信机制的主要优点：**

1. 硬件较简单（多个地址空间，无需考虑紧密耦合关联）
2. 通信是显式的，相应的原语调用：
3. 便于理解：在共享内存模型中，难以知道通信何时发生和结束以及通信的成本。
4. 让程序员关注于通信开销大的并行计算（把重点放在需要去优化的处理开销大的一些方面）

**可在支持上面任何一种通信机制的硬件模型上建立所需的通信模式平台：**

1. 在共享mem（硬件模型）上支持消息传递相对简单（发送一条消息可通过将一部分地址空间的内容复制到另一部分地址空间来实现）
2. 在消息传递的硬件上支持共享存储器就困难的多（因为本身消息传递机制硬件支持很少，它是通过系统调用来完成通信，要想实现共享机制的指令存取很复杂）

**7.1.3并行处理面临的挑战：**

**挑战：**1.程序中有限的并行性+2.相对较高的通信开销（Amdahl定律解释）

**1.有限的并行性使机器要达到好的加速比十分困难（书P261例题上）**

**2.多处理机中远程访问的较大延迟（典型50-10000个时钟周期）（书P261例题下）**

**问题的解决：**

1. **并行性不足：**通过采用并行性更好的算法来解决
2. **远程访问延迟的降低：**靠体系结构支持和编程技术

在并行处理中，负载平衡、同步和存储器访问延迟等影响性能的因素长依赖于高层应用的特点，如应用程序中数据的分配、并行算法的结构以及数据在空间和时间上的访问模式等。

**依据应用程序特点可把多机工作大致分为两类：**

**单个程序**在多处理机上的并行工作负载和**多个程序**在多处理机上的并行工作负载

**重要性能度量——计算与通信的比例（追求高比例）**

计算与通信比例随着处理数据规模的增大而增加，随着处理器数目的增加而降低。用更多处理器求解固定规模大小的问题会导致通信量增加；因此增加处理器的时候应调整问题的规模，从而使通信的时间保持不变。

**7.2对称式（集中式）共享存储器体系结构：**

**7.2.1多处理机（一个共享mem,多个cache）Cache一致性：**

1. 多个处理器共享一个**存储器**。

2. 处理器规模较小，总线带宽有限，这种机器十分经济。

3. 支持对共享数据和私有数据的Cache缓存：私有数据供一个单独的处理器使用，共享数据供多个处理器使用。

4. 共享数据进入Cache产生了一个新的问题：**Cache的一致性问题。**

**（1）不一致产生的原因（Cache一致性问题）**

I/O操作：Cache中的内容可能与I/O系统输入输出形成的存储器对应部分的内容不同。**(cache中内容与mem对应部分的内容不一样)**

共享数据：不同处理器的Cache都保存相应存储器单元的内容**（两个处理器Cache对同一个相应存储器单元操作会有不同的值）**——书P263/表5-1

（2）存储器是一致的（非正式定义）

如果对某个数据项的任何读操作均可得到其最新写入的值，则认为这个存储系统是一致的。

**存储系统行为的两个不同方面：**

**1.返回给读操作是什么值（相关性coherence）**

**2.什么时候才能将已写入的值返回给读操作 （一致性consistency）**

**一致性满足条件：（多处理机需要硬件支持一致性）**

1. 处理器P对X进行一次写之后又对X进行读，读和写之间没有其他处理器对X进行写，则读的返回值总是写进的值。
2. 一个处理器对X进行写之后，另一处理器对X进行读，读和写之间无其它写，则读X的返回值应为写进的值。
3. 对同一单元的写是顺序化的，即任意两个处理器对同一单元的两次写，从所有处理器看来顺序都应是相同的。

假设：直到所有的处理器均看到了写的结果，一次写操作才算完成（并非立即生效）；允许处理器无序读，但是必须按照程序的顺序写。

**7.2.2实现一致性的基本方案：**

在一致的多处理机中，Cache提供两种功能：

1. **共享数据的迁移（把远程拿到本地）：**降低了对远程共享数据的访问延迟。
2. **共享数据的复制（多个节点有同一数据的拷贝）：**不仅降低了访存的延迟，也减少了访问共享数据所产生的冲突

小规模多处理机不是采用软件而是采用硬件实现Cache一致性

1. **Cache一致性协议**：对多个处理器维护一致性的协议。
2. 关键：跟踪共享数据块的状态
3. **共享数据状态跟踪技术：**
4. **目录**（专门的硬件——目录存储器）：**物理存储器中共享数据块的状态及相关信息均被保存在目录中**
5. **监听：**每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。

监听：Cache通常连在共享存储器的总线（数据总线和地址总线）上，各个Cache控制器通过监听总线来判断他们是否有总线上请求的数据块（比较地址等）

**3.两种协议（保证内容新）**

（1）写作废协议：在一个处理器写某个数据项之前保证它对该数据项有唯一的访问权。E.g书P265/表5-2

（2）写更新协议：当一个处理器写某数据项时，通过广播使其他Cache中所有对应的该数据项拷贝进行更新。

**（3）写作废和写更新协议性能上的差别**

1）对**同一数据的多个写而中间无读操作**的情况，写更新协议需进行多次写广播操作，而在写作废协议下只需一次作废操作

2）**对同一块中多个字进行写**，写更新协议对每个字的写均要进行一次广播，而在写作废协议下仅在对**本块第一次写**时进行作废操作

3）从一个处理器写到另一个处理器读之间的延迟通常在写更新模式中较低（因为最新值之前已广播）而在写作废协议中，需要读一个新的拷贝（因为要从存储器，可能还是远程存储器中拿）

因为总线为基础的多处理机带宽有限，所以在基于总线的多处理机中，写作废是大多选择。

**7.2.3监听协议及其实现：**

（1）小规模多处理机中实现写作废协议的关键利用总线进行作废操作，每个块的有效位使作废机制的实现比较容易。

（2）写直达Cache,因为所有写的数据同时被写回主存，则从主存中总可以取到最新的数据值。

（3）对于写回Cache（没有立刻更新）,得到数据的最新值会困难些，因为最新值可能在某个cache中，也可能在主存中。

**（4）在写回Cache条件下的实现技术：**

1）用Cache中块的标志位实现监听过程

2）给每个Cache块**加一个特殊的状态位**说明其是否为共享

3）因为每次总线任务均要检查Cache的地址位，这可能与CPU对Cache的访问冲突（因为一个Cache访问需要监听总线比对地址，CPU也要访问Cache，也要对地址进行操作；一边挂CPU一边挂总线，从而造成冲突）

通过下列两种技术之一降低冲突：

复制标志位（CPU操作一套，总线一套；并行操作）

采用多级包含Cache（每级Cache都有自己的标志位，CPU操作第一级Cache的标识，总线操作三级Cache的标识）

**7.3分布式共享存储器体系结构（通过目录方式）：**

**7.3.1基于目录的Cache一致性**

存储器分布于各节点中，所有的节点通过网络互连。访问可以是本地的，也可以是远程的。

**刚开始硬件无需支持Cache一致性：规定共享数据不进入Cache，仅私有数据才能保存在Cache中**

优点：所需的硬件支持很少（因为远程访问存取量仅是一个字（或双字）而不是一个Cache块）

缺点：（1）实现透明的软件Cache一致性的编译机制能力有限（每次（每个字）都要启动一个远程的访问，延迟大）

（2）没有Cache一致性，机器就不能利用取出同一块中的多个字的开销接近于取一个字的开销这个优点，这是因为共享数据是以Cache块为单位管理。当每次访问要从远程存储器取一个字是，不能有效利用共享数据的空间局部性。

（3）诸如预取等延迟隐藏技术对于多个字的存取更为有效，比如针对一个Cache块的预取。

**因此，分布式共享存储器体系结构现在采用硬件目录解决Cache一致性：**

**1.）目录协议：**

**目录：**用一种专用的存储器所记录的数据结构，它记录着可以进入Cache的每个数据块的访问状态，该块在各个处理器的共享状态以及是否修改过（共享状态是否写读）等信息。

**2）对每个节点增加目录表后的分布式存储器的系统结构（书P283/图5-14）**

**（1）目录协议的基本点**

1.在每个节点增加了目录存储器用于存放目录。

2.存储器的每一块在目录中对应有一项。

3.每一个目录项主要有状态（目录对应存储块当前状态：有无该块，块是否共享还是独占写状态）和位向量（共有N（处理器的个数）位，每一位对应于一个处理器的局部Cache，用于指出该Cache中有无该存储块的拷贝）两种成分。

**（2）目录必须跟踪每个Cache块的状态：**

**Cache块状态有三种：**

**共享：**在一个或多个处理器上有这个块的拷贝，且主存中的值是最新值（所以Cache均相同）

**未缓冲：**所有处理器的Cache都没有此块的拷贝。

**专有：**仅有一个处理器上有此块的拷贝，且已对此块进行了写操作，而主存的拷贝仍是旧的，这个处理器称为该块的拥有者。

**（3）由于写作废操作的需要，还必须记录共享此块的处理器信息**

**方法：**对每个主存块设置一个位向量。

当此块被共享时，每个位指出与之对应的处理器是否有此块的拷贝。

当此块为专有时，可根据位向量来寻找此块的拥有者

**（4）宿主结点：存放有存储器块和对应地址目录项的结点。**

**7.3.2目录协议及其实现：**

基于目录的协议中，目录承担了一致性协议操作的主要功能。

1. **发往一个目录的消息会产生两种不同类型的操作：**

更新目录状态

发送消息（从而能够）满足**请求服务**

1. **目录项可能接收到三种不同的请求：**

读失效、写失效、数据写回

1. **在各个状态下所接收到的请求和相应的操作：**
2. **当一个块处于未缓冲状态，对此块发出的请求及处理操作为：**

**读失效：**将存储器数据送往请求方处理器，且本处理器成为此块的唯一共享节点，本块状态转为共享。

**写失效：**将存储器数据送往请求方处理器，此块成为专有。

**2）当一个块处于共享状态，mem中的数据是其当前最新值，对此块发出的请求及处理操作为：**

**读失效：**将存储器数据送往请求方处理器，并将其将入共享集合。

**写失效：**将数据送往请求方处理器，对共享集合中所有的处理器发送写作废消息，且将共享集合置为仅含有此处理器，本块状态变为专有

**3）当一个块处于专有状态，本块最新值保存在共享集合指出的拥有者处理器中，对此块发出的请求及处理操作为：**

**读失效：**将“取数据”消息发往拥有者处理器，使该块的状态变为共享，并将数据送回目录结点写入存储器，进而把该数据返送请求方处理器，将请求方处理器加入共享集合。

**写失效：**本块将有一个新的拥有者

**数据写回：**拥有者处理器的Cache要替换此块时必须将其写回，从而使存储器中有最新拷贝（宿主结点实际上成为拥有者），此块成为非共享，共享集合为空。

**如此多操作，开销代价大**

1. **对基于目录的Cache一致性的多种改进（降低向量的长度）：**

有限映射目录（假设实际应用中可以共享最大的处理器的数目）

链式结构目录（共享一个链上加一个）

1. **基于目录的Cache一致性协议是完全由硬件实现的；此外，还可以用软硬结合的办法实现。**

**7.5同步：**

**7.5.1基本硬件原语——必须保证硬件执行读和写过程的原子性**

**1.原子交换（Atomic Exchange）：将一个存储单元的值和一个寄存器的值进行交换。**（如果存储单元对应于一个同步锁，假设0代表锁可用，1代表锁不可用；交换后可以表示存储单元的锁被寄存器所对应的进程占用）——耗时间

**2.测试并置（test\_and\_set）：先测试一个值，如果符合条件则修改其值。**（存储单元里对应的锁是0，**读出来（测试）**发现是0**（满足条件）**，空闲，就占用这个锁；然后+1，**修改**其值）。

**3.读取并加1（fetch\_and\_increment）：返回存储单元的值并自动增加该值**（读出来0自动加1）

**4.指令对LL&SC（两条指令，先链取，再条件存——成功返回1，失败返回0）：从第二条指令的返回值可以判断该指令对的执行是否成功。（**取是0，占用锁后为1 ，根据1判断是否成功**）**

**前三个指令可能是两次操作（读一次，写一次），而第四个分成两条指令，这两条指令间不能插入对存储单元进行操作（写）的指令**

书P289/例子（指令对实现同步单元同步锁的例子）

**7.5.2 用一致性实现锁**

采用多处理机的一致性机制来实现**旋转锁**（spin locks）。

**旋转锁：**处理器环绕一个锁不停地旋转（测试）而请求获得该锁。当锁的占用时间很少以及加锁过程延迟很低时可采用旋转锁。（这里只要敢占这个锁，同步锁在这里同步的处理机进程数量很多的时候（自旋锁阻塞处理器，在循环中等待锁被释放），旋转锁开销会很大）

**通过缓存实现一致性**：把锁从远程拿到进程的cache中，cache操作这个锁就像它在操作自己的局部数据一样，获得高速的操作时间

**通过一致性实现锁的两点好处：**

1.可使环绕的进程对本地cache块进行操作

2.可利用锁访问的局部性（上次使用了一个锁的处理器，不久再次用到（可将锁值主存在这个处理器的缓存中））