体系结构 第四章

by xjc

第四章 指令级并行

4.1 指令级并行的概念

定义：

- 几乎所有的处理机都利用*流水线来使指令重叠并行执行*，以达到提高性能的目的。
- 这种指令之间存在的*潜在并行性*称为指令级并行。

4.1.1 流水线处理机的实际CPI（了解）

实际CPI：实际CPI=理想CPI+各类停顿的CPI

$$CPI_{流水线} = CPI_{理想} + 停顿_{结构冲突} + 停顿_{数据冲突} + 停顿_{控制冲突}$$

理想CPI：理想CPI时衡量流水线最高性能的一个指标
IPC：Instructions Per Cycle，每时钟周期完成的指令条数

4.1.2 基本程序块（了解）

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

4.1.3 循环级并行（4.5具体了解）

定义：使一个循环中的不同循环体并行执行。
原因：循环体中存在的并行性最常见且最基本
研究重点：每一次循环都可以与其他的循环重叠并行执行；在每一次循环的内部，却没有任何的并行性
开发并行性技术：

- 循环展开技术
- 向量指令和向量表示

4.1.4 相关与流水线冲突（第三章涉及，去看之前的就行）

（1）相关

类型：数据相关、名相关、控制相关
特点：

- 相关是*程序固有的一种属性*，反映了程序中指令之间的*相互依赖*关系。
- 具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿，则是*流水线的属性*
  解决相关问题的方法：
- 代码调度：保持相关，避免发生冲突
- 代码变换（寄存器重命名）：消除相关

（2）冲突

定义：流水线中由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行。
类型：结构冲突、数据冲突、控制冲突

4.1.5 正确执行程序的两个属性

关键属性：数据流、异常行为

（1）保持异常行为

定义：改变*指令执行顺序*，程序中*会发生的异常*不变。
保持方法：保持程序的*数据相关和控制相关*即可实现
特殊（了解）：有时，不遵守控制相关既不影响异常行为，也不改变数据流。可以大胆地进行指令调度，把失败分支中的指令调度到分支指令之前。

（2）数据流

定义：指数据值从其**产生者指令**到其**消费者指令**的实际流动

- 分支指令使得数据流具有*动态性*，因为它使得给定指令的数据可以有多个来源。
- 仅仅保持*数据相关性*是不够的，只有再加上保持*控制顺序*，才能够保持程序顺序

4.5 循环展开和指令调度

4.5.1 循环展开和指令调度的基本方法

目的：充分开发指令之间存在的*并行性*，找出不相关的指令序列，让它们在流水线上*重叠并行执行*
增加并行性简单常用的方法：

- 开发*循环级并行性*;
- 循环展开后，*重命名+指令调度*开发更多并行性

指令调度的限制特征：

1. 程序*固有的指令级并行性*
2. 流水线功能部件的*执行延迟*

例题：*请看例题4.6，指令调度和循环展开的基本方法*

循环展开技术：

- 把循环体的代码复制多次并按顺序排列，然后相应调整循环的结束条件。
- 这给编译器进行指令调度带来了更大的空间

4.2 指令的动态调度

4.2.0 静态调度和动态调度

指令的静态调度（编译器）：

原理：

- 使用*编译器*，在*编译期间进行代码调度和优化*，以减少相关和冲突。
- 通过把*相关的指令拉开距离*来减少可能产生的停顿（编译器分开相关指令）

指令的动态调度（硬件）：

- 在程序的执行过程中，依靠*专门硬件*对代码进行调度，减少*数据相关*导致的停顿
- 优点：
  - 能够处理在*编译时情况不明的相关*，并简化了编译器；
  - 能够使本来是面向某一流水线优化编译的代码在*其他的流水线（动态调度）上也能高效*地执行。
- 缺点：以硬件复杂性的显著增加为代价

4.2.1 动态调度的基本思想

（1）基本思想和概念

使用流水线的局限性：指令必须按序流出和执行，指令相关性导致流水线停滞

➢ 考虑下面一段代码：

```
DIV.D    F4，F0，F2
SUB.D    F10，F4，F6
ADD.D    F12，F6，F14
```

SUB.D指令与DIV.D指令关于F4相关，导致流水线停顿。

ADD.D指令与流水线中的任何指令都没有关系，但也因此受阻。

解决方法： 允许乱序执行

具体解决步骤

1. 译码（ID）划分为*流出*（IS，检查结构冲突）和*读操作数*（RO，等待数据冲突消失后读操作数）两个阶段
2. 乱序执行可能发生*WAR和WAW冲突*，使用*Tomasulo算法寄存器重命名*消除
3. *多条指令同时执行：* 需要多个功能部件/流水部件
4. 乱序带来的问题：
   - *异常处理复杂*（精确异常处理、不精确异常处理）
   - *动态调度要保持正常的异常行为*
   - *不精确异常*（指令i发生异常时，处理机的状态与严格按照程序顺序执行时i的状态不同）

（2）动态调度算法

动态调度算法：记分牌算法、Tomasulo算法

记分牌算法：

1. **目标：** 在资源充足时，尽可能早地执行没有数据阻塞的指令，达到每个时钟周期执行一条指令
2. **记录的信息**：指令状态表、功能部件状态表、寄存器状态表

4.2.2 Tomasulo算法（主要是计算题实现，记忆其次）

（1）基本概念

核心思想：

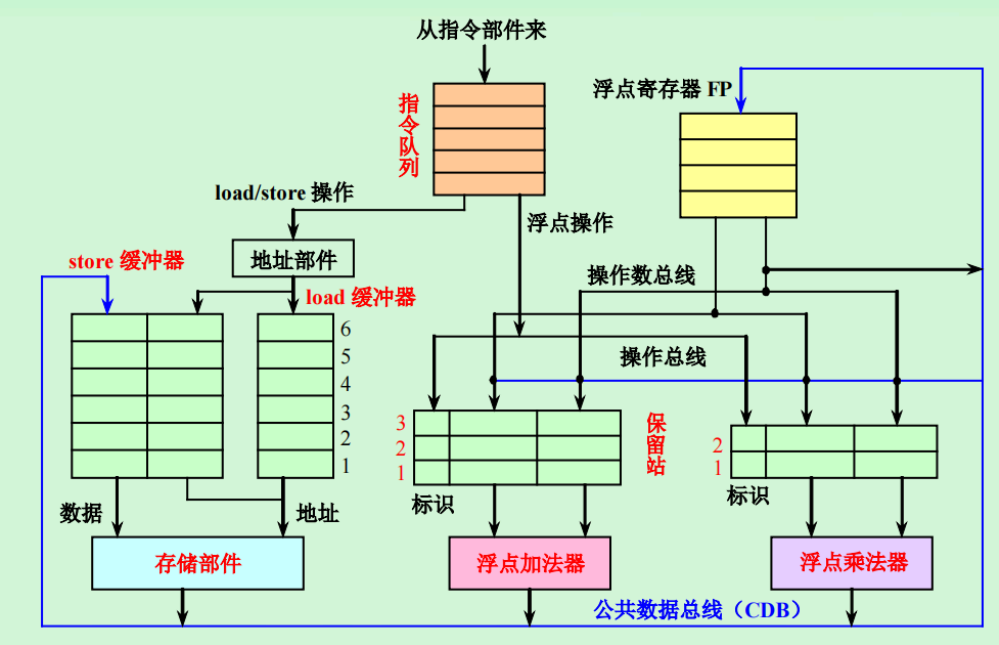
- *记录和检测指令相关*，操作数一旦就绪就立即执行（是不是有点类似记分牌），把发生RAW冲突（没读就写了的可能性最小，因为立马执行数据相关指令）的可能性减少到最小；

- 通过*寄存器换名*来消除WAR冲突和WAW冲突，使用保留站和流出逻辑完成

基于Tomasulo算法的MIPS处理器浮点部件基本结构（理解）：

保留站的功能：保存一条*已经流出并等待到本功能部件执行*的指令。每个保留站都有一个标识字段，*唯一地标识了该保留站*

公共数据总线CDB：一条重要的数据通路



算法特点（理解）：

1. *冲突检测和指令执行控制*是分布的
2. 计算结果通过CDB直接从产生它的保留站传送到所有需要它的功能部件，而不用经过寄存器 *（保留站→CDB→功能部件）*

优点（理解）：

3. 冲突检测逻辑是分布的
4. 消除了WAW冲突和WAR冲突导致的停顿

需要的信息表（重要记忆）：

5. *指令状态表*

	指　　令	指令状态表		
		流出	执行	写结果
①	L.D　F6 , 34(R2)	√	√	√
②	L.D　F2 , 45(R3)	√	√	
③	MUL.D　F0 , F2 , F4	√		
④	SUB.D　F8 , F6 , F2			
⑤	DIV.D　F10 , F0 , F6			
⑥	ADD.D　F6 , F8 , F2			

6. *保留站*

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						

8. 每个保留站有以下几个字段：
   - Op：要对源操作数进行的操作。
   - Qj，Qk：将产生源操作数的保留站号。
     - 等于0表示操作数已经就绪且在Vj或Vk中，或者不需要操作数。
   - Vj，Vk：源操作数的值。
     - 对于每一个操作数来说，V或Q字段只有一个有效。
     - 对于load来说，Vk字段用于保存偏移量。
   - Busy：为“yes”表示本保留站或缓冲单元“忙”。
   - A：仅load和store缓冲器有该字段。开始是存放指令中的立即数字段，地址计算后存放有效地址。

7. *寄存器状态表*

	寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	0	0	0	0	ADD1		…	

（2）算法实现 非常重要！

请看例题4.1和4.2

指令		指令状态表		
		流出	执行	写结果
L. D	F6,34(R2)	√	√	√
L. D	F2,45(R3)	√	√	√
MUL. D	F0,F2,F4	√	√	
SUB. D	F8,F6,F2	√	√	√
DIV. D	F10,F0,F6	√		
ADD. D	F6,F8,F2	√	√	√

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL. D	Mem[45+Regs[R3]]	Regs[F4]			
Mult2	yes	DIV. D		MEM[34+Regs[R2]]	Mult1		

域	寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1					Mult2		

图 4.4　MUL. D 指令准备写结果时各状态表的内容

4.3 动态预测分支技术

4.3.0 基本概念

1. 定义：
   - 根据*分支指令过去的表现*来预测其未来的行为
   - 有更好的预测准确度和适应性
2. 影响因素：
   - 预测的*准确性*
   - 预测正确和不正确两种情况下的*分支开销*
     - 决定分支开销的因素：流水线的结构、预测的方法、预测错误时的恢复策略等
3. 目的：
   - *预测分支是否成功*
   - 尽快找到*分支目标地址*（或指令）,避免控制相关造成流水线停顿
4. 关键问题（理解）
   - 如何记录分支的*历史信息*
   - 如何根据这些信息来*预测分支的去向*（甚至取到指令）
   - 在预测错误时，要*作废*已经预取和分析的指令，恢复现场，并从另一条分支路径*重新取指令*

4.3.1 采用分支历史表BHT

定义：**分支历史表BHT**（Branch History Table）或**分支预测缓冲器（Branch Prediction Buffer）**

- 最简单的方法。
- 用BHT来记录 *分支指令 最近一次或几次的执行情况（成功或不成功）*，并据此进行预测

使用方法：

1. 只有*1个预测位*的分支预测缓冲：只记录最近1次的历史
2. 采用*2位二进制位*来记录历史：提高预测准确度，
   - 研究证明2位和n位效果差不多；
   - 1次预测失败仍然保留状态，2次预测失败则变换

使用条件（理解）：判定分支是否成功所需的时间 大于 确定分支目标地址所需的时间

实现方法：BHT可以存放在*指令Cache*中，也可以用*一个专门的硬件*来实现

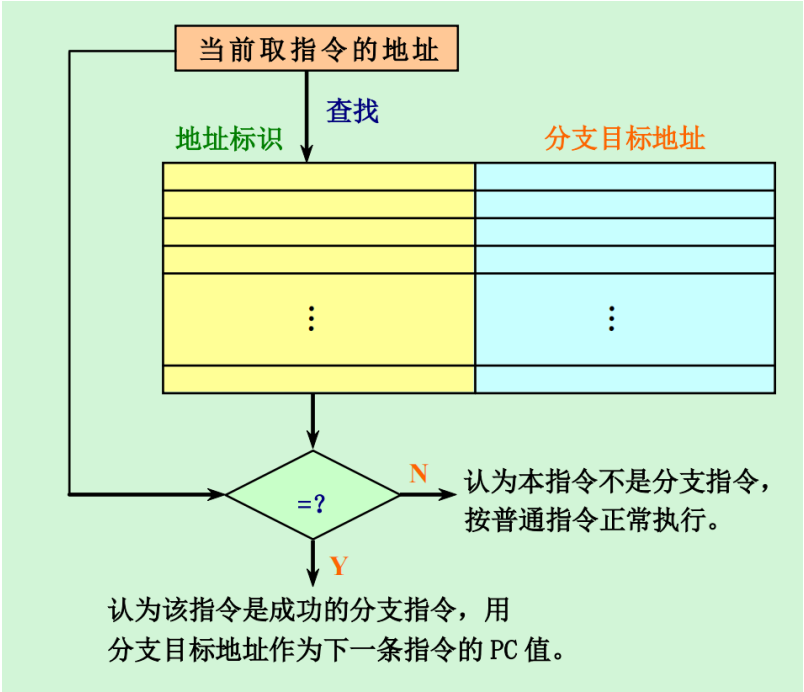
4.3.2 采用分支目标缓冲器BTB

目标：将分支的开销降为0

方法：

- 存放方式：将*分支成功的指令的地址*和它的*分支目标地址*都放到**分支目标缓冲器（BTB）**中保存起来
- 查找标识：缓冲区以*分支成功的指令的地址*作为标识

  内容：
- 看成是用专门的硬件实现的一张表格。
- 表格中的每一项至少有两个字段：（类似key-value键值对）
  - *分支成功的指令的地址（作为标识）*
  - *对应的分支目标地址*



BTB改进版：存放*一条或者多条*分支目标处的指令

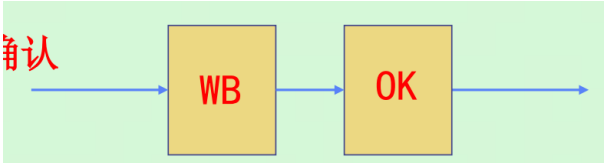
- 取指令速度更快
- 多流出
- 可进行分支折叠优化（零延迟无条件分支）

4.3.3 基于硬件的前瞻执行ROB

1. 基本思想：预测分支指令结果，*始终假设猜测正确，并继续执行后续指令*。指令的结果暂存到*ROB（ReOrderBuffer）的缓冲器*中。待执行确认后，*才将结果写入寄存器或存储器*

   缝合的思想：
   - 动态分支预测。
   - 前瞻地执行后续指令。
   - 动态调度
2. 关键思想：允许指令乱序执行，但必须顺序确认
3. 对Tomasulo算法的修改

   把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：**写结果**和**指令确认**

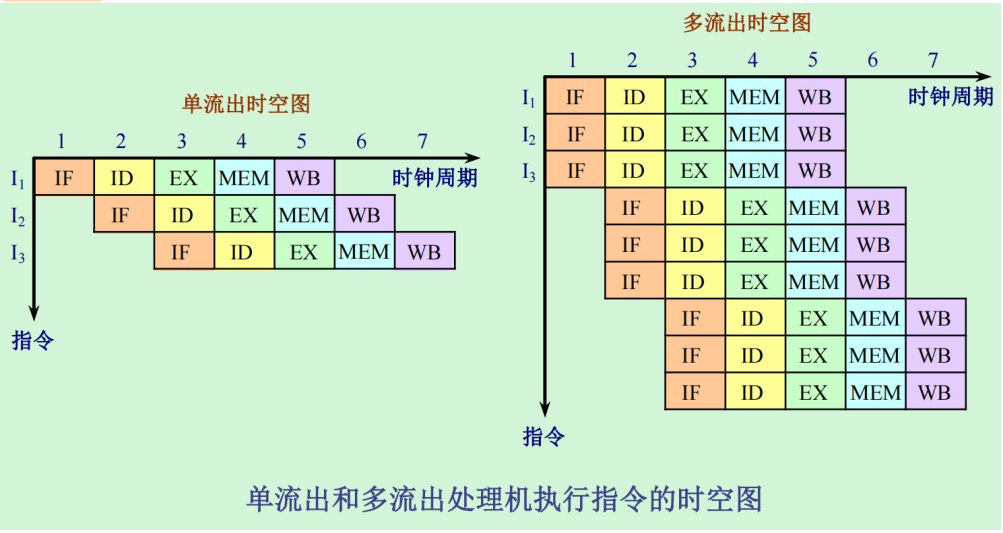
   

4. **优缺点**
   - 通过ROB实现了*指令的顺序完成*。
   - 能够*实现精确异常*。
   - 很容易地*推广到整数寄存器和整数功能单元*上。
   - **主要缺点**：所需的硬件太复杂。

4.4 多指令流出技术

4.4.0 基本概念

定义： 一个时钟周期内流出多条指令，CPI<1



单流出和多流出处理机执行指令的时空图

多流出指令机的三种基本风格： (注意区分)

1. 超标量（Superscalar）
   - **定义：** 有多条流水线的处理机
   - **n流出：** 每个始终周期流出n条指令，就称为n流出处理机。
   - **调度方式：** 动静结合
     - 编译器进行静态调度
     - 基于Tomasulo算法进行动态调度
2. 超长指令字VLIW（Very Long Instruction Word）
   - **指令字：** 把并行执行的多条指令*组装成一条很长的指令字*，并控制多个执行部件工作
   - **指令条数：** 每个时钟周期流出的*指令条数是固定的*，这些指令构成一条*长指令*或者一个*指令包*。
   - **并行性的显式：** 指令包中，指令之间的并行性是通过*指令显式地表示出来*的。
   - **指令调度：** 由*编译器静态完成*的。
3. 超流水线处理机（Superpiple）
   - **定义/原理：** 流水线中使*各流水段时间小于1个时钟周期*，从而在1个周期内*分时流出多条指令*，称为超流水线处理机。
   - **1/n流出：** 实际上n条指令不是同时流出的，而是*每隔1/n个周期流出一条指令*。
   - **流水线周期：** 实际上流水线周期为*1/n个时钟周期*。
   - **其他定义（了解）：** 在有的资料上，把指令流水线级数为*8或8以上的流水线处理机*称为超流水线处理机。
   - **典型的超流水线处理器：** SGI公司的MIPS系列R4000

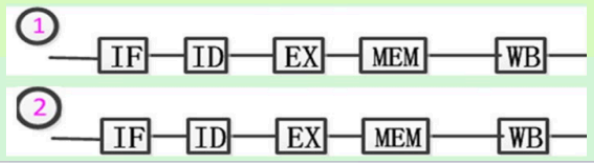
4. 超标量处理机比超长指令字的优势（只需要记忆简单的）
   - 超标量结构对程序员是透明的
   - 没有经过编译器针对超标量结构优化的代码也可以运行
   - 效果不会很好。要想达到很好的效果方法之一：使用*动态超标量调度技术*

技 术	流出结构	冲突检测	调 度	主要特点	处理机实例
超标量（静态）	动态	硬件	静态	顺序执行	Sun UltraSPARCII/III
超标量（动态）	动态	硬件	动态	部分乱序执行	IBM Power2
超标量（猜测）	动态	硬件	带有猜测的动态执行	带有猜测的乱序执行	Pentium III/4, MIPS R10K, Alpha 21264, HP PA 8500, IBM RS64III
VLIW /LIW	静态	软件	静态	流出包之间没有冲突	Trimedia，i860
EPIC	主要是静态	主要是软件	主要是静态	相关性被编译器显式地标记出来	Itanium

4.4.1 基于静态调度的多流出技术（双流出）

1. 要求：同时取两条指令（64位），译码两条指令（64位）
2. 指令处理步骤：
   - Cache中取2条指令
   - 确定指令是否可以流出（0~2）
   - 发送到对应功能部件
3. ==指令执行过程 ==

指令类型	流水线工作情况							
整数指令	IF	ID	EX	MEM	WB			
浮点指令	IF	ID	EX	EX	MEM	WB		
整数指令		IF	ID	EX	MEM	WB		
浮点指令		IF	ID	EX	EX	MEM	WB	
整数指令			IF	ID	EX	MEM	WB	
浮点指令			IF	ID	EX	EX	MEM	WB
整数指令				IF	ID	EX	MEM	WB
浮点指令				IF	ID	EX	EX	MEM



4. 硬件优化方式：”1条整数+1条浮点“并行流出，*需要增加的硬件很少*

5. load/store浮点寄存器访问冲突：他们使用整数部件，会增加对浮点寄存器的访问冲突，解决方法是增设一个浮点寄存器的读/写端口。
6. 定向路径增设：由于流水线中的指令多了一倍，定向路径也要增加
7. 性能障碍：
    - load指令：load后续3条指令都不能使用其结果，否则会引起停顿
    - 分支延迟：如果分支指令是流出包中的第一条指令，则其延迟是3条指令；否则就是流出包中的第二条指令，其延迟就是两条指令

4.4.2 基于动态调度的多流出技术（不知道考不考，不背了）

扩展Tomasulo算法：支持两路超标量

（1）两路指令

- 每个时钟周期流出两条指令；
- 一条是整数指令，另一条是浮点指令
- 指令按顺序流向保留站，否则会破坏程序语义。
- 将整数所用的表结构与浮点用的表结构分离开，分别进行处理，这样就可以同时地流出一条浮点指令和一条整数指令到各自的保留站

（2）实现多流出的两种方法

关键：对保留站的分配和对流水线控制表格的修改
方法一：在*半个时钟周期里完成流出步骤*，这样一个时钟周期就能处理两条指令
方法二：设置*一次能同时处理两条指令*的逻辑电路

（3）例题

第9次课p48，Tomasulo超标量处理
双流出动态调度流水线的性能受限原因：

1. 整数部件和浮点部件的*工作负载不平衡*，没有充分发挥出浮点部件的作用。应该设法减少循环中整数型指令的数量。
2. 每个*循环迭代中的控制开销太大*。5条指令中有两条指令是辅助指令。应该设法减少或消除这些指令。
3. *控制相关*使得处理机必须等到分支指令的结果出来后才能开始下一条L.D指令的执行。

4.4.3 超长指令字技术（VLIW）

原理：把能并行执行的多条指令组装成一条很长的指令
指令字：指令字被分割成*一些字段*，每个字段称为一个*操作槽*，直接*独立地控制一个功能部件*。
调度方式：均由编译器完成
例题：第9次课p55 例4.5
存在的问题：

- **程序代码长度增加了：**
  - 解决：
    - 指令共享立即数字段
    - 指令压缩存储
    - 调入Cache或译码时展开
- **采用了锁步机制：**任何一个操作部件出现停顿时，整个处理机都要停顿。
- **机器代码的不兼容性**

4.4.4 多流出处理机的受限因素

1. 程序所*固有的指令级并行性*。
2. *硬件实现上的困难*。
3. 超标量和超长指令字处理器固有的*技术限制*。

4.4.5 超流水线处理机

1. 定义/原理：流水线中使*各流水段时间小于1个时钟周期*，从而在1个周期内*分时流出多条指令*，称为超流水线处理机
2. 指令流出/指令周期：1/n周期
3. 典型的超流水线处理器：SGI公司的MIPS系列R4000