Lecture 9 流水线

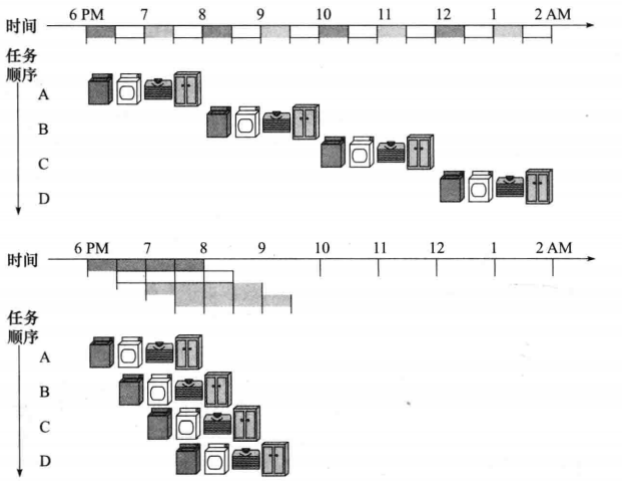
1. 流水线概述

介绍

流水线 piplining 是一种实现多条指令重叠执行的技术，与生产流水线类似

流水线对于单个任务来说，**过程总处理时间并没有缩短**，然而在有多批任务时流水线之所以快的原因是因为所有的工作都在**并行进行**，因此单位时间能够完成的工作大大增加，实际是改善了**吞吐率**

如果所有步骤所需的时间一样，并且有足够的工作可做，那么流水线得到的速度提高倍数等于流水线中步骤的数目



处理器中的流水线方式

MIP 指令的五个步骤

一个 MIPS 指令通常包含如下 5 个步骤

1. 从指令存储器中读取指令
2. 指令译码的同时读取寄存器，MIPS 的指令格式允许同时进行指令译码和读寄存器
3. 执行操作或计算地址

4. 从数据存储器中读取操作数
5. 将结果写回寄存器

流水线的五级

一个流水线 CPU 通常包含五个流水级

1. IF：取指阶段（其方框表示指令的存储器）
2. ID：指令的译码或寄存器的读取阶段（虚线方框表示要读取的寄存器堆）
3. EX：指令的执行阶段（外边图形表示 ALU）
4. MEM：存储器访问阶段（其方框表示数据存储器）
5. WB：写回阶段（虚线方框表示被写回的寄存器堆）

例题1 单周期和流水线的时钟周期

下面比较流水线指令执行与单周期指令执行的平均执行时间，在单周期模型中，所有的指令的执行都花费一个时钟周期，假设主要的功能单元的操作时间为

- 存储器访问：200ps
- ALU操作：200ps
- 寄存器堆的读写：100ps

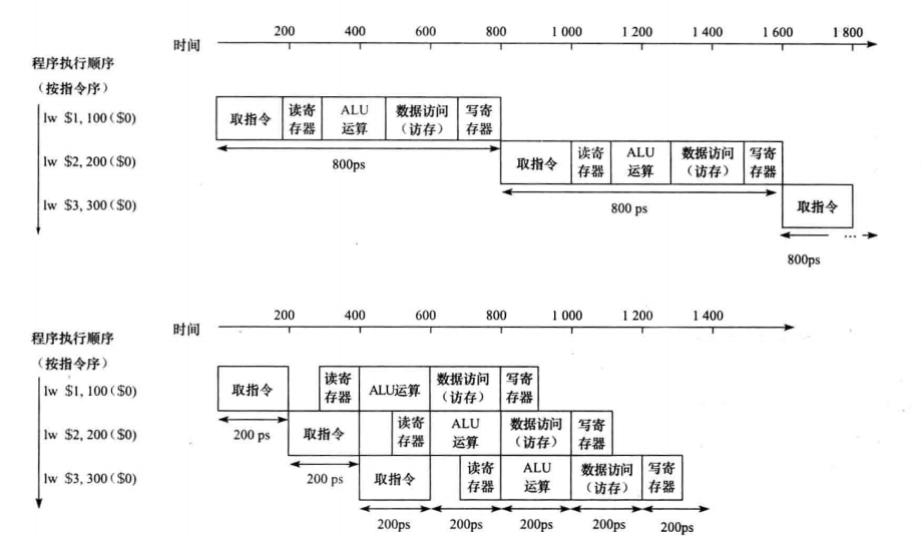
分别说明单周期和流水线下，CPU 的时钟周期

我们以 lw、sw、add、sub、AND、OR、slt 和 beq 为例

指令类型	取指令	读寄存器	ALU 操作	数据存取	写寄存器	总时间
取字（lw）	200ps	100ps	200ps	200ps	100ps	800ps
存字（sw）	200ps	100ps	200ps	200ps		700ps
R 型（add、sub、AND、OR、slt）	200ps	100ps	200ps		100ps	600ps
分支（beq）	200ps	100ps	200ps			500ps

时钟周期必须满足最慢的指令，因此在现在的条件下，每一条指令所需要的执行时间为 800ps

下图比较了三条 lw 指令非流水线和流水线方式的执行过程



所有的流水级 **pipline stage** 都只花费一个时钟周期的时间，**因此时钟周期必须能满足最慢操作的执行需要**，但周期模型下时钟周期为 800ps，在流水线执行模型下时钟周期为 200ps（而不是有些步骤可以达到的100 ps）

把上面讨论的流水线模型能够获得的加速性能比归纳成

如果流水线各阶段操作是平衡的 balanced

那么流水线机器上的指令执行时间为（理想）

$$Time\ between\ instruction_{pipline} = \frac{Time\ between\ instruction_{nonpiplined}}{Number\ of\ pipe\ stages}$$

- 流水线所带来的性能提高是通过增加指令的吞吐率 **throughput**
- 而不是减少单条指令的执行时间 latency

MIPS 面向流水线的指令集设计

MIPS 指令集有一些适合流水线设计的好处

- 所有指令都是 32-bit 的
    - 更容易取指 IF 和解码 ID 在一个周期完成
- 指令格式少，且正则
    - 更容易解码和读寄存器 ID 在一步完成
- 加载 / 存储地址
    - 可以在第三级 EX 计算地址，在第四级 MEM 访问数据存储器
- 存储器操作数对齐
    - 存储器访问在一个周期完成

2. 流水线冒险

有些情况下，在下一个时钟周期中下一条指令不能执行，这种情况成为冒险 hazard

阻塞 stall / bubble

正式的叫法是**流水线阻塞 pipline stall**，它经常被称为**气泡 bubble**

结构冒险 Structure hazard

硬件不支持多条指令在同一时钟周期执行，需要的资源繁忙，使用资源而发生冲突

如果 MIPS 流水线只有一个存储器（数据和指令共用一个），那么

- 加载 / 存储需要数据访问
- 为了取指会阻塞 stall 该时钟周期
    - 可能导致流水线气泡 bubble

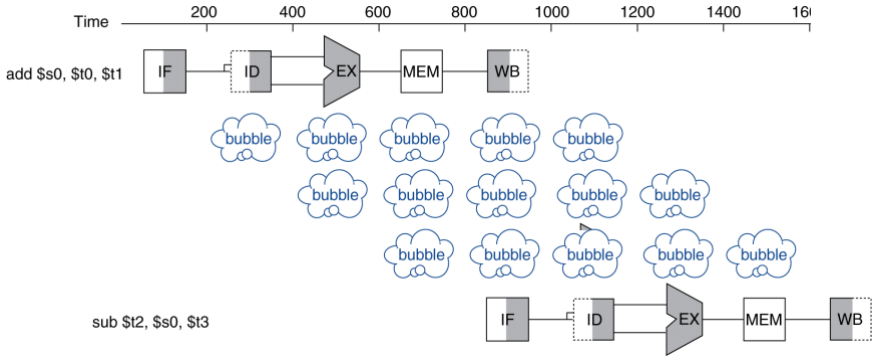
因此，流水线数据路径需要单独的指令 / 数据存储器

数据冒险 Data hazard / Pipline data hazard

由于一条指令必须等待另一条指令的完成而造成流水线暂停，即因无法提供指令执行所需要的数据而导致指令不能在预定的时钟周期内完成的情况

两条指令间的旁路

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



减法指令需要使用加法指令的和 $s0，在不做任何干涉的情况下，这一数据冒险会很严重地阻碍流水线

- 加法指令到第五级才会写回结果
- 然后减法指令才能开始计算

这意味着在流水线中浪费了 3 个时钟周期

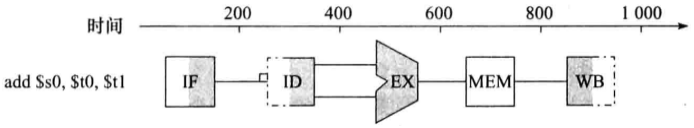
前推 Forwarding / 旁路 Bypassing

一个基本的解决方法是，**在解决数据冒险问题之前不需要等待指令的执行结束**，对于上述例子，一旦 ALU 产生了加法运算的结果，就可以将它用于减法运算的一个输入项

前推 forwarding 或者旁路 bypassing

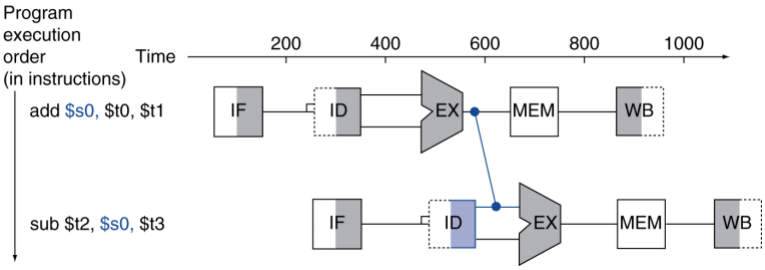
- 从内部资源中直接提前得到缺少的运算项的过程
- 从内部寄存器而非程序员可见的寄存器或存储器中提前取出数据

指令流水线的图形表示如下



对于上述指令（add $s0, $s0, $t1）来说

- 阴影表示该资源被指令所使用
- 左半边阴影表示它们在此步骤中被写入
- 右半边阴影表示它们在此步骤中被读取

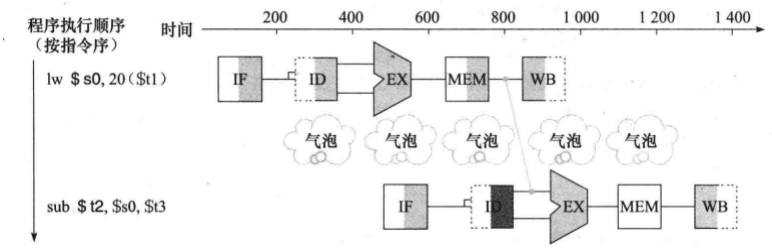


- 图中的连接表示从 add 指令的 EX 操作输出到 sub 指令的 EX 操作输入的**旁路路径 bypassing line**，从而替换掉 sub 的第二步从寄存器 $s0 读取的值

取数-使用型数据冒险 load-use data hazard

如果将上述指令替换成

```
lw $s0, 20($t1)
sub $t2, $s0, $t3
```



- 由于数据间的依赖，所需要的数据只有在前一条指令流水线的第四级完成之后才能生效
- 这对于 sub 指令的第三级输入来说太迟了
- 此时即使采用了旁路机制，在遇到**取数-使用型数据冒险 load-use data hazard** 时，流水线不得不阻塞一个步骤

例题2 重新安排代码以避免流水线阻塞

考虑下面的 C 代码

```
a = b + e;
c = b + f;
```

和对应的 MIPS 指令

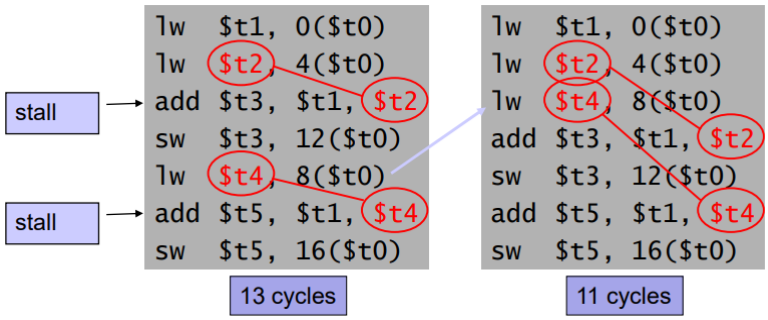
```
lw  $t1, 0($t0)     # b
lw  $t2, 4($t0)     # e
add $t3, $t1, $t2   # a = b + e
sw  $t3, 12($t0)    # save a
lw  $t4, 8($t0)     # f
add $t5, $t1, $t4   # c = b + r
sw  $t5, 16($t0)    # save c
```

试找出上述代码段存在的冒险并且试着重新安排指令顺序以避免流水线阻塞

可以看出，两条 add 指令都存在冒险，因为它们都依赖于上一条 lw 指令，解决方案可以如下

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0) # 将这条指令移到上面
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

在一个具有旁路功能的流水线处理器中，执行这个重新排序后的指令要比上面的指令快 2 个时钟周期



控制冒险 Control hazard / Branch hazard

决策依赖于一条指令的结果，而其他指令都在执行，因为取到的指令并不是所需要的（或者说指令地址的变化并不是流水线所预期的），而导致指令不能再预定的时钟周期执行

在 MIPS 流水线中

- 需要在流水线更早的时候比较寄存器和计算目标地址
- 可以增加硬件使得它在流水线的第二级 ID 便可以计算出跳转地址

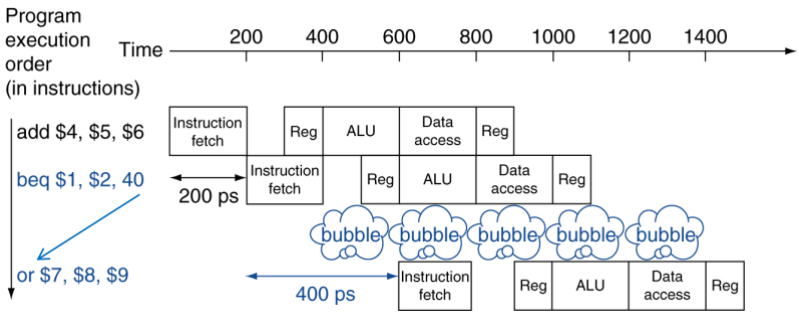
阻塞 stall

计算机中的决策是**分支指令**，注意到，在取分支指令后，紧跟着会取下一条指令，但是分支指令不一定会按顺序跳转到下一条指令

一种可能的解决方法是取到分支指令后**立即阻塞流水线**，直到流水线确定分支指令的结果并知道下一条真正在哪里为止

如果 MIPS 增加硬件使得它在流水线的第二级 ID 便可以计算出跳转地址，包含条件分支的流水线执行情况就如下图所示

假设分支执行后跳转到 or 指令，不跳转则执行 lw 指令



不论如何，在分支指令之后都会**插入一个周期的流水线阻塞**，或者叫做气泡

如果不能在第二级解决分支问题，那么分支结构上的阻塞将导致更大的速度下降

例题3 阻塞对分支性能的影响

评价分支阻塞对单位指令时钟周期数 CPI 的影响，假设其它所有的指令的 CPI 为 1

在 SPECint 2006 中，分支指令约占执行指令的 17%，由于其它指令的 CPI 都为 1，而分支指令阻塞需要多一个时钟周期，因此平均 CPI 为 1.17，与理想情况相比，速度下降了 1.17 倍

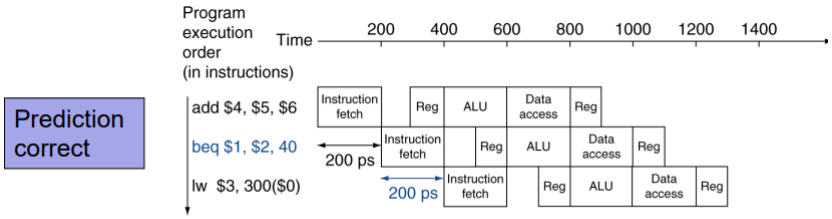
预测 predict

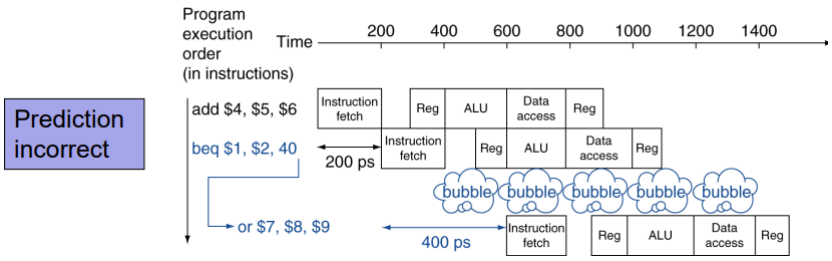
分支预测 branch predition：一种解决分支冒险的方法，它预测分支结果并且立即沿预测方向执行，而不是等真正的分支结果确定后才开始执行，当预测错误时，流水线控制必须确保被错误预测的分支后面的指令执行不会生效，并且必须在正确的分支地址处重新开始流水线

计算机是采用**预测**的方法来处理分支的，一种简单的预测方法就是总是预测分支未发生，当预测正确时，流水线还会全速执行，只有当分支法神时流水线才会阻塞

在 MIPS 流水线中

- 简单预测分支不会发生
- 没有延迟，直接取分支指令的下一条指令





- 预测成功：流水线正常执行，执行 lw 指令
- 预测失败：流水线发生一次阻塞，跳转到 or 指令

更加成熟的分支预测与处理

更加成熟的分支预测预测一些分支发生，另一些不发生

静态分支预测 Static branch prediction

- 基于典型的分支行为
- 循环体底部的分支总是会跳回循环体的顶部，在这种情况下，由于分支总发生并且向前跳转，因此我们可以预测分支回跳转到前面的某一地址处

动态分支预测 Dynamic branch prediction

- 保存每次分支的历史记录，然后利用这个历史记录来预测，当历史记录的数量和类型足够多时，这种动态分支预测的方式能够达到 90% 的正确率
- 硬件度量实际的分支行为
    - 记录每个分支跳转历史情况
- 假设未来的跳转行为将继续这种情况
    - 当错误时，阻塞，并更新历史记录

延迟分支 delayed branch

在 MIPS 体系结构中，延迟分支顺序执行下一条指令，在一条指令延迟之后再开始执行分支，在一条指令延迟之后再开始执行分支，由于编译器会自动排列指令，使分支的行为达到程序员的要求，MIPS 编译器会在延迟分支指令的后面紧跟着一条不受该分支影响的指令，发生了的分支会改变这条安全指令后的指令地址