西安电子科技大学
人工智能学院

计算机组成与体系结构

第7章 流水线技术与指令级并行

本章第3次课重点

- 流水线性能提高
  - ✓结构冒险
  - ✓控制冒险

流水线中基本性能问题

7.5.1 流水线的基本性能问题

- 理想的CPU体系结构是它的每个部分都一直在“工作”，且每个时钟周期至少启动/完成一条新的指令，这种结构的最佳实现之一就是流水线。
- 合理地划分、设计流水线，使流水线每段内的操作必须在一个时钟周期内完成，就有可能实现每条流水线*CPI*＝1的理想目标。

7.5.1 流水线的基本性能问题

- 限制指令流水线性能提高的因素：
  - ✓ 流水线的深度受限于流水线的延迟、流水线段的时间不均衡和流水线的额外开销。
  - ✓ 指令执行时可能存在的相关（dependence）或“冒险（hazard）”问题。
- 冒险：指相邻或相近的两条指令因存在某种关联，后一条指令不能在原指定的时钟周期开始执行。
- 冒险或相关有3类：
  - ✓ 结构（structural）冒险：资源冲突
  - ✓ 数据（data）冒险：
    一条指令需要用到前面某条指令的结果
  - ✓ 控制（control）冒险：
    分支等转移类指令/其他能够改变PC值的指令

7.5.1 流水线的基本性能问题

- 冒险可能引起流水线停顿（stall），停顿也称为流水线气泡（pipeline bubble）。
- 有停顿的流水线的*CPI*为：

  CPI = 理想流水线CPI+(结构冒险停顿 + 数据冒险停顿 + 控制冒险停顿) / 指令数

- 有停顿的流水线的加速比为：

$$\text{加速比} = \frac{\text{流水线深度}}{1 + \text{平均每条指令的流水线 停顿周期}}$$

- 消除冒险、减少停顿是提高已有流水线性能最重要的手段。

流水线性能提高

——结构冒险及解决

7.5.2 结构冒险

有两种情形会导致结构冒险：

- 部分功能单元没有充分流水。
  解决办法：将流水线设计的更合理。
- 资源冲突（resource conflicts）：当两个以上流水线段需要同时使用同一个硬件资源时，发生冲突。
  解决方法：

  MIPS/RISC-V

  - ✓ 增加资源副本。
  - ✓ 改变资源以便它们能并发的使用。
    - 不相关的数据尽量使用不同的寄存器
    - 寄存器重命名
  - ✓ 通过延迟（或暂停）流水线的冲突段或在冲突段插入流水线气泡（气泡在流水线中只占资源不做实际操作），使各段“轮流”使用资源。

流水线性能提高

——数据冒险及解决

7.5.3 数据冒险

- 指令在流水线中重叠执行有可能改变指令读/写操作数的顺序
- 当一条指令的结果还未有效生成，该结果就被作为后续指令的操作数时，数据冒险出现。

例7.8 某程序中包括如下两条指令：

I1：sub x2, x3, x4 // (x2)=(x3)-(x4)

I2：add x10, x2, x5 // (x10)=(x2)+(x5)

这两条相邻指令被送入图7.18所示的RISC-V流水线上。执行前，各寄存器中的值分别为(x2)=20，(x3)=15，(x4)=10，(x5)=5，(x10)=0。两条指令执行后，寄存器x10的值为多少？

解：I1指令执行完时，应有(x2)=(x3)-(x4)=15-10=5。

但I2指令在ID段读取x2时，I1指令还未进入WB段，所以I2指令读取的x2不是I1指令的结果，而是I1指令执行前的值，所以I2指令执行后，实际为(x10)=(x2)+(x5)=20+5=25

而原程序预期是(x10)=(x2)+(x5)=5+5=10。

- 结论：因为I1指令的目的操作数和I2指令的第一操作数发生相关，并在流水线上产生了冒险，所以指令执行结果出现错误。

7.5.3 数据冒险

- 冒险类型：假设两条指令i和j，i先进入流水线。
  - 写后读(RAW)：在i写入结果之前，j先去读。
  - 读后写(WAR)：在i读操作数之前，j先写入进行了修改。
  - 写后写(WAW)：在i写入结果之前，j先写入结果。

例7.9　4级指令流水线，各级分别为取指IF、读数RD、执行EX和写结果WB。简单指令（第1、2、4、6条指令）在EX1段执行，需1个时钟周期；复杂指令（第3、5条指令）在EX2段执行，需3个时钟周期。

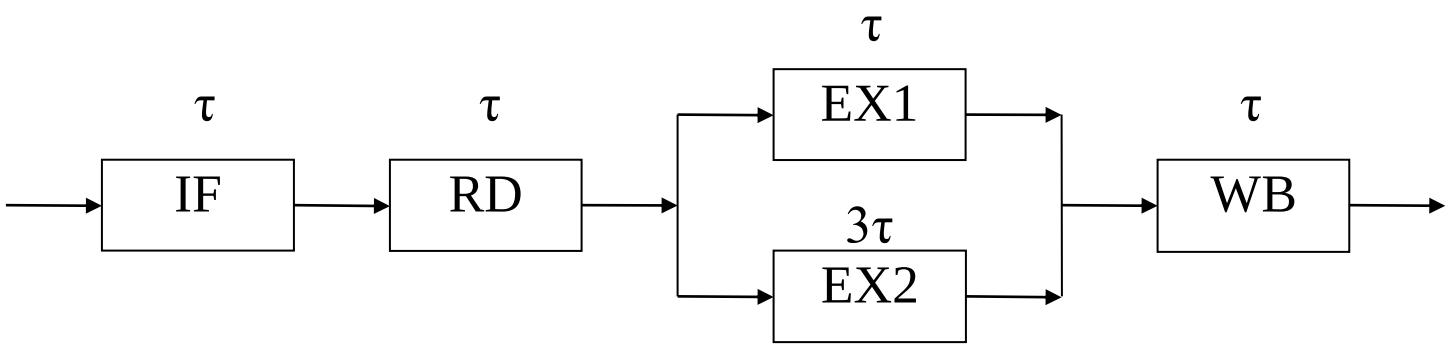


图7.24 流水线结构示意图

7.5.3 数据冒险

表7.5 典型的数据冒险类型

指令	1	2	3	4	5	6	7	8	9	10	冒险类型
1. R1+R2→R0	IF	RD	EX1	WB							RAW
2. R0-R3→R4		IF	RD	EX 1	WB						
3. R0→10H(R5)			IF	RD	EX 2-1	EX 2-2	EX 2-3	WB			WAR
4. R6→R0				IF	RD	EX 1	WB				
5. R1×R2→R0					IF	RD	EX 2-1	EX 2-2	EX 2-3	WB	WAW
6. R1+R2→R0						IF	RD	EX 1	WB		

冒险造成指令2、3和6的结果错误。RAW是真数据冒险。

7.5.3 数据冒险

➢ 解决办法：

- ✓ 采用直通/转发（forwarding）技术　　图7.25

  控制逻辑将前面指令的结果从其产生的地方直接连通到当前指令所处的位置。

- ✓ 增加专用硬件

  增加流水线互锁（pipeline interlock）硬件。互锁硬件先要检测流水线中指令的数据相关性，当互锁硬件发现数据相关时，使流水线工作停顿下来，直到冒险消失为止。

- ✓ 利用编译器

  流水线调度/指令调度：编译器可以对指令重新排序或插入空操作指令，使得加载任何冲突数据的操作被延迟，但对程序逻辑或输出不受影响。

- ✓ 对寄存器读写做特别设计

  时钟周期前半部分写，后半部分读，解决RAW冒险。

流水线性能提高

——控制冒险及解决

7.5.4 控制冒险

- 使程序执行顺序发生改变的转移指令有两类：
  - ✓ 无条件转移指令（如无条件跳转、调用、返回指令等）
    - 某些CPU（如UltraSPARC III）：紧跟在无条件转移指令之后的指令必须执行。
    - 另一些CPU：采取相对复杂的方法，如提前计算出转移目标地址。

跳转示例

  - ✓ 条件分支转移指令（为零跳转、循环控制指令等）
    - 不仅需要延迟槽，而且一直到流水线的深处，取指单元才能知道到哪里去取下一条指令。
      - 分支延迟槽（branch delay slot）：程序中位于转移指令后面的存储单元位置。
    - 条件分支指令对流水线性能的影响远比无条件转移指令要大。

对条件分支指令的处理方法：

7.5.4 控制冒险 — 对条件分支指令的处理方法

7.5.4.1 冻结（freeze）流水线

- 早期的流水线CPU采用最简单的冻结（freeze）流水线的方法处理分支指令。
- 即一旦在指令译码段检测到分支指令，就在转移目标地址确定之前保存或删除所有紧随分支指令之后的指令。
- 当分支指令从执行段流出、确定出新的PC值时，流水线才继续依据新PC值填充流水线。
- 这样会严重地影响流水线的性能。

7.5.4 控制冒险 — 对条件分支指令的处理方法

7.5.4.2 静态分支预测（static branch prediction）

- 可以采用的预测方法：
  - ✓ 预测分支不会发生（predict never taken）
    “出错检测处理”
  - ✓ 预测分支总是发生（predict always taken）
    “循环”
  - ✓ 由编译器预测
    确定次数循环
- 预测错误时，如何取消已经执行和将要执行的指令：
  - ✓ 执行，结果保存在临时寄存器中。
  - ✓ 将要被覆盖的寄存器的原值保存在临时寄存器中。
- 适合级数较小的流水线

7.5.4 控制冒险 — 对条件分支指令的处理方法

7.5.4.3 动态分支预测（dynamic branch prediction）

- 通过记录分支指令的近期运行历史，并以此作为预测的依据，来提高分支预测的准确度。
- 分支历史表（branch history table）

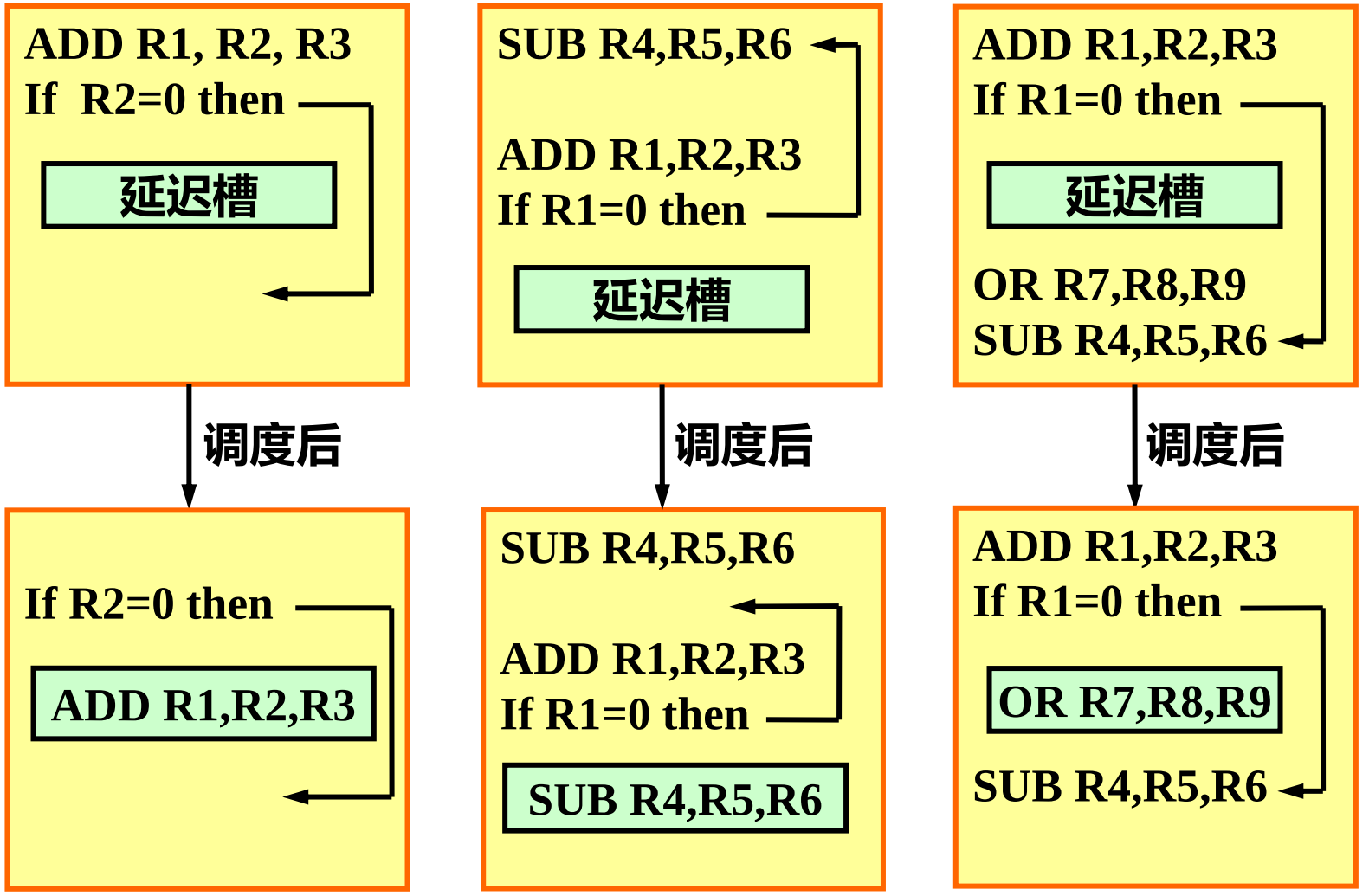
  又称分支预测缓存（branch-prediction buffer）
  - ✓ 1位分支历史表 图7.26(a)
  - ✓ 2位分支历史表 图7.26(b)
  - ✓ 分支目标缓存（branch target buffer，BTB）

    如果历史表中表明地址X处的分支指令上次发生了跳转并转移到了地址Y处，而且现在的预测仍然是“转移”，那么就假定这次将再次转移到地址Y处。用于Pentium。 图7.26(c)
- 加入全局分支信息的预测器
  - ✓ 相关预测（correlating predictor）
  - ✓ 双级预测（two-level predictor）
  - ✓ 锦标赛分支预测器（tournament branch predictor）

7.5.4 控制冒险 — 对条件分支指令的处理方法

7.5.4.4 延迟分支（delayed branch）/延迟转移

- 利用编译器对指令代码进行重新排序，并插入有用指令或空（NOP）操作指令。
- 流水线遇到分支指令时，按正常方式处理，同时执行延迟槽中的指令。
- 编译器的任务就是在延迟槽中放入有用的指令，称为延迟槽调度。有三种调度方法：
  - 从分支前（from before）调入
  - 从目标处（from target）调入
  - 从失败处（from fall-through）调入
- 采用延迟分支法的限制：
  - 放入延迟槽的指令需要满足一定的条件
  - 编译器要有预测分支是否成功的能力



(a) 从分支前
(被调度的指令必须与分支无关)

(b) 从目标处
(必须保证在分支失败时执行被调度的指令不会导致错误。有可能需要复制指令)

(c) 从分支失败处
(必须保证在分支成功时执行被调度的指令不会导致错误)

图7.28 调度延迟槽的方法

7.5.4 控制冒险 — 对条件分支指令的处理方法

例7.7 假设某流水线在分支转移成功后会导致3个时钟周期的停顿，若分支指令的频度为30%，理想*CPI*=1，则实际*CPI*为多少？

解：实际*CPI* =1+30%×3≈2，因此实际的加速比将只能达到理想加速比的50%。

例7.11 一个有10轮循环的循环程序段，假设采用分支预测缓冲器（1位分支历史表）进行是否继续循环的预测，则该循环分支的预测准确度为多少？

解：若该循环程序段仅执行一次，且之前已将预测位设置为跳转，则仅在最后一轮循环中，对循环分支不可避免地会做出错误预测（退出循环时不需要跳转）。因为循环分支有9次正确预测，1次错误预测，预测准确率为90%。

若该循环程序段执行多次，则之后的每次循环程序段执行时，由于上次循环结束时预测错误，且已将预测位修改为不跳转，使得进入本循环程序段的第一轮循环就出现错误预测。然后翻转预测位，使后8轮正确预测，最后一轮又出现错误预测。因此，此时循环分支的预测准确率仅为80%。

- 控制冒险对流水线性能造成的损失远比数据冒险要大得多。