计算机组成
(2022秋)

计算机组成课程组
（刘旭东、高小鹏、肖利民、栾钟治、万寒）
北京航空航天大学计算机学院中德所
栾钟治

➢1

习题5——单周期处理器

- 已发布
  - Spoc平台
- 11月18日截止
  - 23:55
- 在sopc提交
  - 电子版，可手写

➢2

回顾：异常和中断

- “突发的” 事件需要改变控制流
  - 不同的指令集体系结构会使用不同的术语
- *异常*
  - CPU内部产生，一旦检测出来需要“立即”处理
- *中断*
  - 来自外部I/O控制器，“方便”的时候处理（除非高优先级）
- 需要牺牲性能

➢3

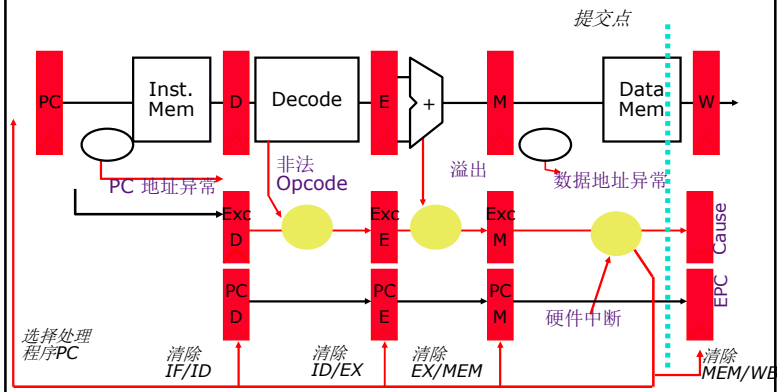
回顾：处理异常

- 可重启的异常
- MIPS中异常由系统控制协处理器（CP0）处理
  - 保存出问题（或者被中断）的指令的PC内容
  - 保存问题的描述
  - 转跳到异常处理代码
  - 通知操作系统
- 指令：MFC0、MTC0
  - MFC0：读取CP0寄存器至通用寄存器
  - MTC0：通用寄存器值写入CP0寄存器

寄存器号	寄存器
12	SR
13	CAUSE
14	EPC
15	PRID

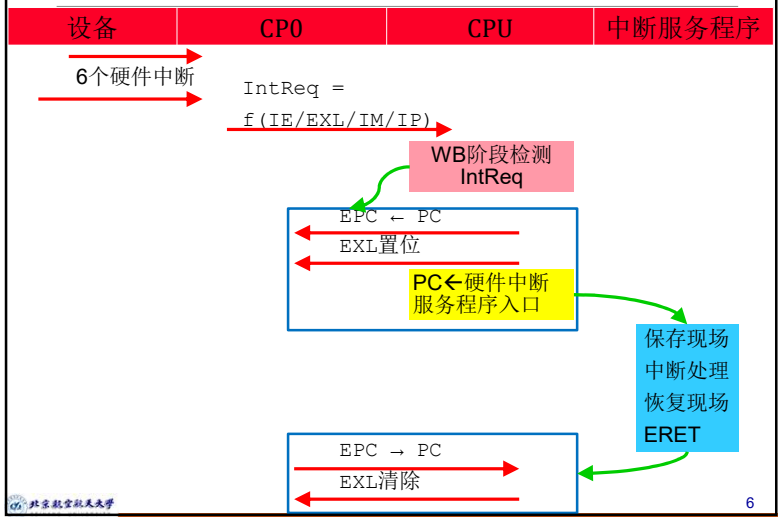
➢4

回顾：异常处理(5段流水线)



➢5

回顾：中断响应机制分析：软硬件协同



➢6

回顾：精确异常

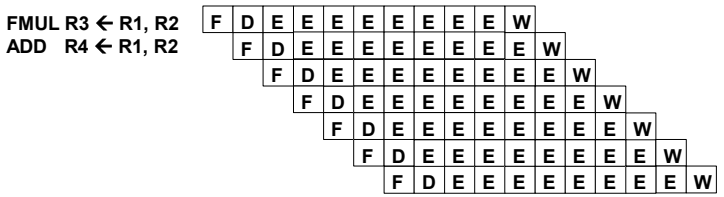
- 流水线的收益来自于指令执行开销的重叠
- 许多的异常必须是精确的
  - 所有在出错指令之前的指令必须完成
  - 出错指令和它之后的所有指令必须被销毁（清空），不得提交结果
  - 异常处理程序必须开始执行

➢7

回顾：多周期“执行”

- 不是所有指令的“执行”时间都是一样长的
- 思路: 有多个不同的功能单元，会花费不同数量的时钟周期
- 程序序的保证和精确异常

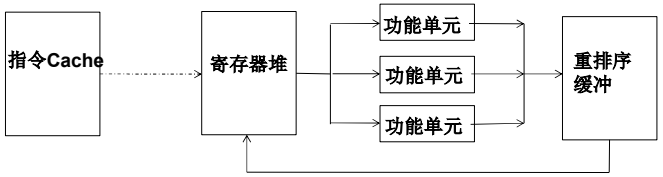
- 流水线中确保精确异常
  - 思路: 使每个操作花同样的时间



➢8

回顾：解决方案 I: 重排序缓冲 (ROB)

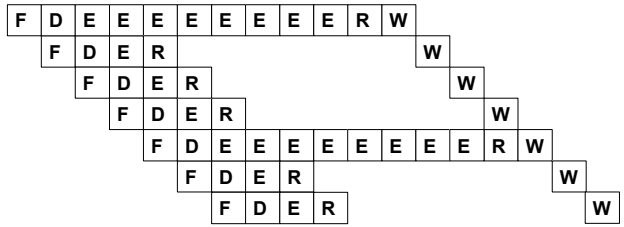
- 思路: 乱序执行指令, 产生体系结构状态可见的结果之前重排序
- 当指令译码时在ROB中预留一个条目
- 当指令执行完时，将结果写入ROB 中相应条目
- 当指令成为ROB中最旧的一条，并且已经执行完（没有异常）时，将结果移动到寄存器堆或者存储器



➢9

ROB: 独立操作

- 结果首先写入ROB, 在指令提交时写入寄存器堆

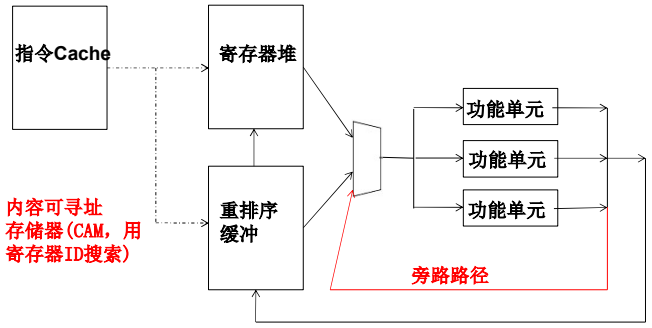


- 如果一个后续的操作需要的值在ROB中会怎么样?
  - 读寄存器堆的同时读ROB。 如何做到?

➢10

ROB: 如何访问?

- 一个寄存器的值可以在寄存器堆里，也可以在ROB中 (还可以在旁路/转发路径上)



➢11

简化ROB访问

- 思路: 使用间接寻址的思想
- 先访问寄存器堆
  - 如果寄存器无效，寄存器堆存储ROB中包含（或将要包含）寄存器值的条目ID
  - 将寄存器映射到ROB条目
- 再访问ROB

还能再简化吗?

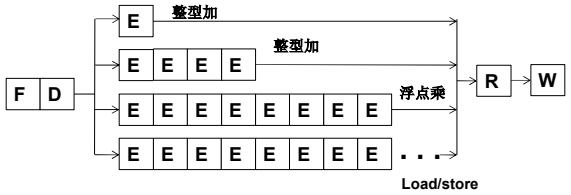
用ROB为寄存器重命名

- 将要保存结果的寄存器根据ROB条目重命名
  - 寄存器ID → ROB条目ID
  - 体系结构寄存器ID → 物理寄存器ID
  - 重命名后, ROB 条目ID被用来指向寄存器
- 这将消除输出相关和反相关
  - 造成一个有大量寄存器的假象

➢12

使用ROB的按序“执行”流水线

- 译码 (D): 访问寄存器堆/ROB, 分配ROB条目, 检查指令是否可以执行, 是则分发指令
- 执行 (E):指令可以完全的乱序执行
- 完成 (R): 将结果写入ROB
- 回收/提交 (W): 检查异常; 如果没有, 将结果写入体系结构寄存器堆或存储器; 如果有, 清空流水线并启动异常处理程序
- 按序分发/执行, 乱序完成, 按序回收



➢13

ROB的Tradeoff

- 好处
  - 用很简单的概念来支持精确异常
  - 可以消除“虚假的”相关
- 坏处
  - 有可能需要访问ROB以获得尚未写入寄存器堆的结果
    - CAM 或间接寻址→ 增加延时和复杂性
- 消除上述缺陷的解决方案
  - 历史缓冲
  - 未来寄存器堆

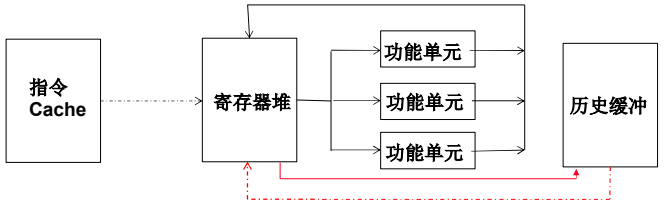
➢14

解决方案 II: 历史缓冲 (HB)

- 思路: 指令执行完成后更新寄存器堆, 但是当有异常发生时撤销那些更新（UNDO）
- 当指令译码时，预留一个HB条目
- 当指令执行完毕时，将目标地址中的旧值存在HB里
- 当指令是HB中最旧的一条并且没有发生异常/中断，丢弃该HB条目
- 当指令是HB中最旧的一条并且有异常需要处理，将HB中保存的旧值依次写回到体系结构状态

➢15

历史缓冲



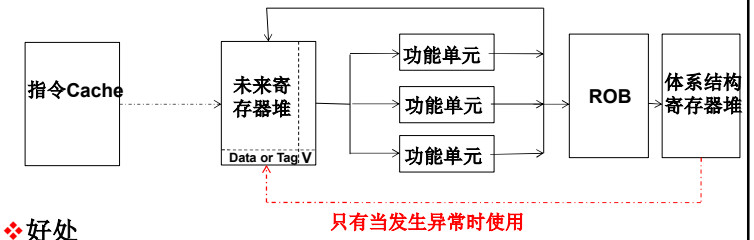
- 好处:
  - 寄存器堆中保有最新的值，HB的访问不在关键路径上
- 坏处:
  - 需要读目的寄存器的旧值
  - 在异常时需要回滚HB→ 增加异常/中断的处理时延

➢16

解决方案 III: 未来寄存器堆(FF) + ROB

- 思路: 维护两个寄存器堆 (投机的和体系结构的)
  - 体系结构的寄存器堆: 按程序序更新以获得精确异常
    - 使用ROB来保证按序的更新
  - 未来的寄存器堆: 一条指令执行完毕后立即更新(如果这条指令是最新的一条写寄存器堆的指令)
- FF 用于快速访问最近的寄存器值(投机的状态)
  - 前端寄存器堆
- 体系结构寄存器堆用于当发生异常时的状态恢复 (体系结构状态)
  - 后端寄存器堆

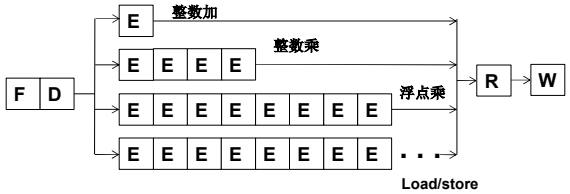
未来寄存器堆



- 好处
  - 不需要从ROB中读取值 (不需要CAM或者间接寻址)
- 坏处
  - 多个寄存器堆
  - 发生异常时需要从一个堆向另一个堆复制数据

有FF和ROB的按序“执行”流水线

- 译码 (D): 访问FF, 在ROB中分配条目, 检查指令是否可以执行，是则分发指令
- 执行 (E): 指令可以完全的乱序执行
- 完成 (R): 将结果写入ROB和FF
- 回收/提交 (W): 检查异常; 如果没有, 将结果写入体系结构寄存器堆或者存储器; 如果有, 清空流水线并启动异常处理程序
- 按序分发/执行, 乱序完成, 按序回收



流水线中检查和处理异常

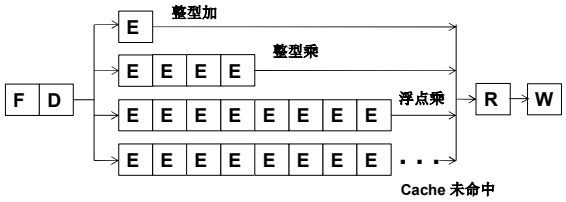
- 当最旧的准备回收的指令被检查出导致了异常, 控制逻辑
  - 恢复体系结构状态(寄存器堆, IP/PC, 存储器)
  - 清空流水线中所有比该指令新的指令
  - 保存IP/PC和寄存器堆 (ISA指定)
  - 重定向取指引擎指向异常处理子程序
    - 异常向量

寄存器 vs. 存储器

- 存储器会怎么样?
- 寄存器和存储器之间有什么根本的不同?
  - 寄存器的相关是静态可知的 – 存储器的相关是动态决定的
  - 寄存器的状态空间小 – 存储器的状态空间大
  - 寄存器状态对其它线程/处理器不可见 – 存储器状态在线程/处理器之间是共享的 (共享存储多处理器)

乱序执行 (动态指令调度)

按序执行的流水线



- 问题: 一个真正的数据相关会使流水线停止分发新的指令到功能(执行)单元
- 分发: 将指令送入功能单元的动作

能不能更好?

- 下面两段代码有何共同之处?

IMUL R3 ← R1, R2	LD R3 ← R1 (0)
ADD R3 ← R3, R1	ADD R3 ← R3, R1
ADD R1 ← R6, R7	ADD R1 ← R6, R7
IMUL R5 ← R6, R8	IMUL R5 ← R6, R8
ADD R7 ← R3, R5	ADD R7 ← R3, R5

- 答案: 第一个ADD指令使整个流水线停顿!
  - 由于源寄存器不可用使得ADD不能分发
  - 后续的独立指令也无法执行
- 上面两段代码的区别是什么?
  - 答案: Load 延迟是不确定的(在运行时才知道)

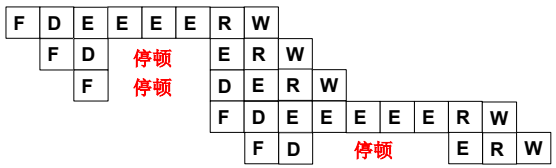
乱序执行(动态调度)

- 思路: 让相关的指令离独立的指令远一点（分离）
  - 相关指令的“休息区”: 保留站
- 监控休息区中每条指令的源“值”
- 当一条指令的所有源“值”可用，则分发该指令执行
  - 指令按数据流序 (非控制流) 分发
- 好处:
  - 延迟容忍: 允许独立指令在有长延迟操作时执行和完成

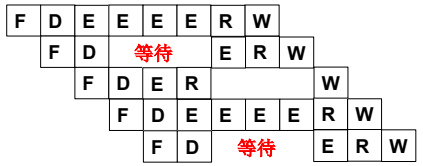
➢25

按序 vs. 乱序分发

- 按序分发+ 精确异常:



- 乱序分发+ 精确异常:



- 16 vs. 12 时钟周期

➢26

乱序执行的支撑条件

1. 需要在“值”的生产者和消费者之间建立连接
   - 寄存器重命名: 给每一个“值”关联一个“标签(tag)”
2. 需要缓冲指令直到它们准备好执行
   - 寄存器重命名之后向保留站插入指令
3. 指令需要持续跟踪源“值”是否可读
   - 当“值”被生产出来，广播“标签(tag)”
   - 保留站中的指令对比自己的“源标签”和广播标签 → 如果匹配, 则源值准备好
4. 当一条指令的所有源值都准备好，需要分发指令到它的功能执行单元 (FU)
   - 当所有源值准备好，指令唤醒
   - 如果有多条指令被唤醒, 需要为每个FU选择一条指令

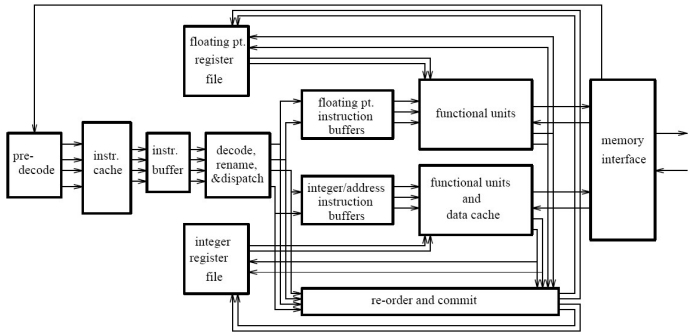
➢27

Tomasulo算法

- Robert Tomasulo发明了带寄存器重命名的乱序执行
  - 用于IBM 360/91的浮点运算单元
  - 阅读: Tomasulo, “An Efficient Algorithm for Exploiting Multiple Arithmetic Units,” IBM Journal of R&D, Jan. 1967.
- 大多数高性能处理器使用的都是它的变种
  - 最初是在 Intel Pentium Pro, AMD K5
  - Alpha 21264, MIPS R10000, IBM POWER5, IBM z196, Oracle UltraSPARC T4, ARM Cortex A15

➢28

乱序执行处理器一般的组织形式



➢29

例子

MUL R3 ← R1, R2
ADD R5 ← R3, R4
ADD R7 ← R2, R6
ADD R10 ← R8, R9
MUL R11 ← R7, R10
ADD R5 ← R5, R11

F	D	E	W

- 假设 ADD (执行阶段4个时钟周期流水), MUL (执行阶段6个时钟周期流水)
- 假设只有一个加法器和乘法器
- 需要多少时钟周期
  - 非流水线机器
  - 非精确异常的按序分发流水线 (无转发和全转发)
  - 非精确异常的乱序分发流水线 (全转发)

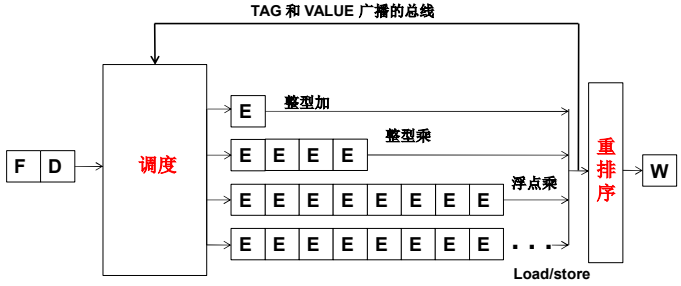
➢30

带精确异常的乱序执行

- 思路: 在指令提交体系结构状态之前利用重排序缓冲对指令进行重排序
- 当指令执行完成时更新寄存器别名表 (本质上是个未来寄存器堆)
- 当指令执行完成并且是最旧的一条指令时更新体系结构寄存器堆

➢31

带精确异常的乱序执行



- 驼峰 1: 保留站(调度窗口)
- 驼峰 2: 重排序(ROB, 又叫指令窗口或者动态窗口)

➢32