系统结构 复习总结

Lecture 01 量化方法

1. 延迟和吞吐量

- 延迟：完成一项任务的时间
- 吞吐量：带宽，单位时间完成的任务数

Lecture 02 ISA

1. ISA分类

- 栈体系架构：操作数隐式位于栈顶
- 累加器架构：操作数隐式为累加器
- 通用寄存器架构：只有显式操作数，要么是寄存器，要么是存储器位置

2. stacking architecture

- 优点
    1. 指令很短
    2. 编译简单
- 缺点
    1. 栈操作很慢
    2. 代码效率太低，很多次swap

3. accumulator

- 优点
    1. 指令很短
    2. 设计很简单
- 缺点
    1. CPI 不同，难以实现流水线

4. GPR

Lecture 03 pipeline

1. 什么是流水线？怎样提高性能？优缺点分析？

1. 多条指令重叠执行的技术
2. 优缺点
    - 优点：提高了指令吞吐量吞吐率(1/CPI)，但不会缩短单条指令的执行时间
    - 缺点：流水线开销，包括：**寄存器延迟和时钟偏差；流水线不平衡：CCT是最慢的一条指令；阻塞**

2. 流水线冒险

主要搞懂：种类、原因、处理方式、举例

1. 结构冒险
    - 原因：硬件资源无法支持所有指令组合；结构冒险时，pp通常停顿一个周期，称为1个**气泡**
    - 处理方式：完全避免结构冒险成本太高，如果sh不多的话，just let it be
    - 举例：前后指令同时访问存储器（存储器只有单读端口）
2. 数据冒险
    - 原因：指令执行存在先后顺序，如果一条指令的执行依赖于先前指令的结果，则存在~
    - 处理方式
        - 转发/旁路：将结果直接传送给需要它的功能单元，eg：EX/MEM->ALU, MEM/WB->ALU
        - 阻塞：适用于LD->ALU（取数->用数型）
        - 增加部件
        - 高级：scoreboarding；renaming
    - 举例：待做
3. 控制冒险
    - 原因：分支指令和其他改变PC的指令实现流水线时可能会导致~
    - 简单处理方式：一旦ID检测到分支跳转，就重新取指，原来的IF阶段相当于一次停顿
    - 降低流水线分支代价
        - 冲刷流水线：保留或删除分支之后的所有指令，直到知道分支结果为止；
        - 预测永远未选中/选中：未选中，继续执行；选中，暂停一个周期，从target开始取指执行，表格要会画
        - 高级：动态分支预测；loop unrolling

3. HW vs SW

1. HW：当数据依赖存在时，由硬件处理，避免阻塞；不改变数据流
    - 简化编译，为某一流水线编译的代码可以在另一个流水线上高效运行
    - 能够处理编译阶段无法检测的依赖：访存指令
    - 是超标量处理器的基础
2. SW：编译器通过separate dependent instructions尽可能减少stall
3. 静态调度优点：

    静态调度缺点：

    动态调度的优点：

    1. 允许针对一种流水线编译的代码在不同流水线上高效执行，不需要再重新编译
    2. 可以处理编译代码时还不能知道相关性的情况
    3. 允许处理器容忍一些预料之外的延迟，它可以在等待解决缺失问题时执行其他代码

    动态调度的缺点：

    硬件的复杂度显著提高

4. timing chart

1. no-bypassing
   - WB阶段能读取，因为前半个周期写，后半个周期读
   - flushing：BNEZ必须等到X的后一个周期才能取下一条指令

Example

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD	R1, 0(R2)	F	D	X	M	W													
DADDI	R1, R1, #1		F	s	s	D	X	M	W										
SD	**R1, 0(R2)**					F	s	s	D	X	M	W							
DADDI	R2, R2, #4								F	D	X	M	W						
DSUB	R4, R3, R2									F	s	s	D	X	M	W			
BNEZ	R4, Loop												F	s	s	D	X	M	W
LD	R1, 0(R2)																	F	D

Since the initial value of R3 is R2 + 396 and equal instance of the loop adds 4 to R2, the total number of iterations is 99. Notice that there are 8 cycles lost to RAW hazards including the branch instruction. Two cycles are lost after the branch because of the instruction flushing. It takes 16 cycles between loop instances; the total number of cycles is $98 \times 16 + 18 = 1584$. The last loop takes two addition cycles since this latency cannot be overlapped with additional loop instances.

2. by-passing+always not taken
   - 数据前推
   - BNEZ不能前推，也就是只有等到上一条指令的W阶段才能读取BNEZ的X指令，**BNEZ的X阶段时下一条指令的F阶段**
3. by-passing+always taken
   - 数据前推
   - BNEZ在D阶段默认跳转，就可以取下一条指令了，也就是说**BNEZ的D阶段就是下一条指令的IF阶段**

Lecure 02 scoreboard

1. 3种相关

1. 数据依赖 **RAW**

   也叫**真数据相关**，传递了3点信息：
   - 冒险的可能性
   - 计算结果必须遵循的顺序
   - 可开发并行度的上限
2. 名称依赖
   - WAW
   - WAR

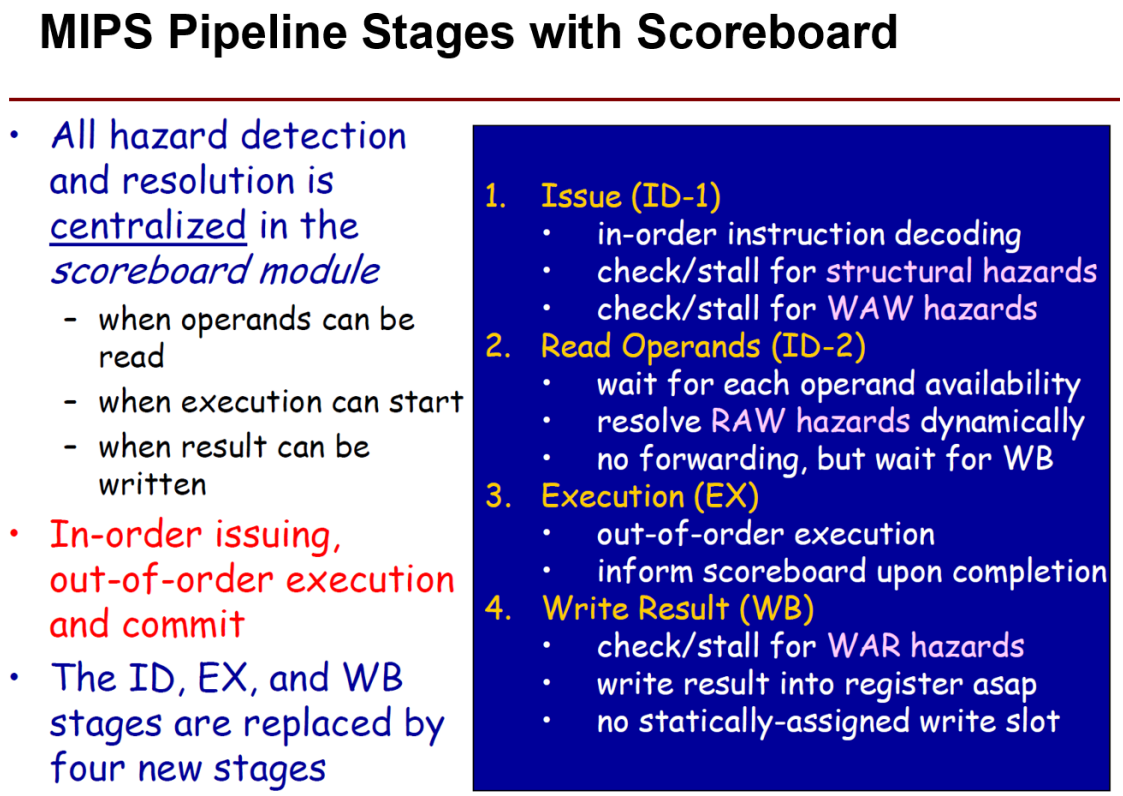
   乱序执行会导致冒险

2. in-order和out-of-order

1. in-order：发射、执行、完成都是按照顺序的，在ID阶段检查结构冒险和数据冒险
2. out-of-order：顺序发射，乱序执行和结束。ID阶段被拆分为：
    - issue：decode，检查结构冒险
    - read operands：直到没有数据冒险

3. scoreboard注意点

1. issue时检查：**FU是否空闲，结构冒险**、目的寄存器是否冲突（看FU表），设置ready标志，设置寄存器状态表（最下面的）；若**WAW**，阻塞issue
2. read-operands：读数据，修改标志；**上周期更新为TRUE的需要等待一周期再读**；没有forwarding
3. excute：注意执行时间
4. wb:wb和更新FU在同一周期内完成；检测**WAR**，如果wb的目的寄存器恰好是某指令**将读未读**（源寄存器，ready=FALSE）的话，必须等待它读了之后的一个周期才能写回



Scoreboard Example:
Clock Cycle = 17

Instruction	Issue	Read Operands	Exec. Complete	Write Result
L.D F6, 34(R2)	1	2	3	4
L.D F2, 45(R3)	5	6	7	8
MUL.D F0, F2, F4	6	9		
SUB.D F8, F6, F2	7	9	11	12
DIV.D F10, F0, F6	8			
ADD.D F6, F8, F2	13	14	16	

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult-1	yes	MUL.D	F0	F2	F4			false	false
Mult-2	no								
Add	yes	ADD.D	F6	F8	F2			false	false
Divide	yes	DIV.D	F10	F0	F6	Mult-1		false	true

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult-1			Add		Divide		

• addition stalls (result is not written) to avoid WAR hazard on F6

3. 循环展开步骤

1. replicate loop body：展开几次（4次），删除前几次的判断指令，即BNEZ
2. merge：合并前几次的循环条件更新的指令，放在最后一次统一更新（反正前面也不会跳转）
3. renaming：对寄存器重命名，消除名字依赖
4. scheduling：调度，调整指令顺序，减少stalls

Lecture 06 Tomasolu

1. Tomasolu 优势（比scoreboard）

1. 更少的结构冒险：两条并行的load
2. 通过广播机制实现从FU到保留站的基于CDB的旁路
3. 清除WAR和WAW阻塞：renaming，不用阻塞

缺点

1. 设计复杂
2. CDB bottleneck，因为CDB必须和所有保留站通信

Tomasulo's Algorithm vs. Scoreboard

	Tomasulo's Algorithm	Scoreboard
introduced with	IBM 360/91 in 1967	CDC 6600 in 1964
resources	3 adders, 2 mul/div, 6 load, 3 store	7 int units, 4 FP units, 5 mem. reference
max window size	14	5
structural hazards	stall issuing	stall issuing
WAR hazards	avoided by register renaming	stall committing
WAW hazards	avoided by register renaming	stall issuing
control	based on reservation stations	scoreboard module
communication	broadcasting to reservation stations (and registers)	buses to/from registers

- With scoreboard both operands are read at once when they are both ready in the register file, while Tomasulo's algorithm reads each operand asap

2. scoreboard和tomasulo对比

scoreboard

scoreboard集中控制

1. 3种冲突
    - WAW：issue阶段检查
    - WAR：wb前检查，若有的话要stall until 成功read
    - RAW：read operands阶段实时监控源寄存器的状态，不过要等一个cycle才能读取

tomasulo

1. 特征
    - 分布式控制，用reservation stations作buffer
    - 用register renaming解决WAW, WAR
    - tracking operands to solve RAW
    - reservation station 比 register 多，解决名称依赖（compile无法解决）

Lecture 07 cache

1. 局部性原理 简答题

1. 时间局部性：如果一个数据被访问，在不久的将来它可能再次被访问
2. 空间局部性：如果一个数据项被访问，与它地址相邻的数据项可能很快也将被访问

> 1. cache既体现了时间局部性（最近用过的数据放在cache，提供更快的访问速度）和空间局部性（数据缺失时，从内存中取出连续的块而不只是一个块）

> 2. 双层循环的外部循环体现了空间局部性（a[1],a[2]...），内部循环体现了时间局部性（每个迭代都用到了a[i]）

2. 写操作

1. 写直达：写通过/写穿，写操作总是同时更新cache和下一存储层次，以保持二者一致性。

   写缺失：要写的数据不在cache中，先从主存中取出块中的字放在cache中，然后将引起缺失的字重新写入cache中。

   写缓冲：一个保存等待写入主存数据的缓冲队列，当把数据写入cache和缓冲队列后，处理器继续执行，等到数据写入主存，缓冲队列可以释放。

   优点

   - 更好实现
   - cache永远clean，处理写缺失更快，因为不用将cache写回到内存
   - 内存和cache永远是一致的
2. 写回：写操作仅将新值写入cache中，只有当被修改过的块被替换时才写到较低层次的存储器中。

   优点

   - 以cache速度写回，很快
   - 对内存中一个快的多个字写仅需要一次写操作

3. cache组织方式 优缺点

1. 直接映射
   - 简单、开销不大（tag很短）
   - 如果反复访问两个冲突的块，缺失代价很大
2. 全相联
   - 决定块放在哪个位置很灵活
   - 要依次检查tag，很耗时
3. 组相联

   集合了上面的优点

4. 写缺失

1. 写分配：数据块先从主存中取到cache，再该块的恰当区域写数据
2. 写不分配：只更新主存中块的一部分，而不写入cache中（因为有时候会写一整个块，所以不需要写分配）