计算机组成
(2022秋)

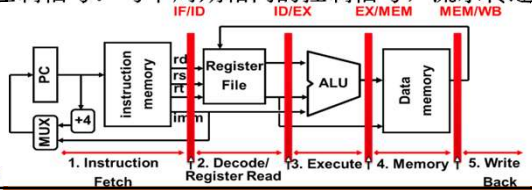
计算机组成课程组
（刘旭东、高小鹏、肖利民、栾钟治、万寒）
北京航空航天大学计算机学院中德所
栾钟治

习题5——单周期处理器

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

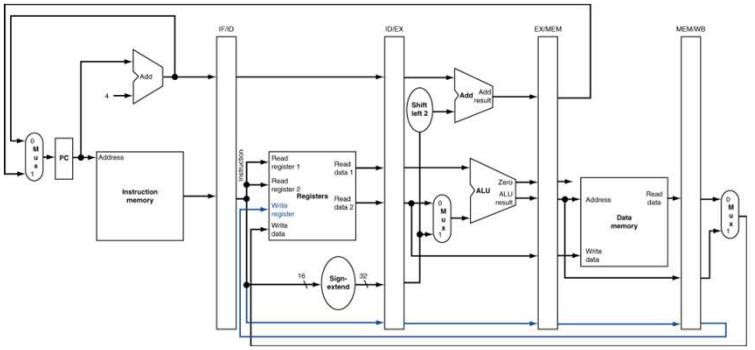
回顾：MIPS流水线设计基本思路

- 数据通路与单周期相同
  - 划分为5个流水段：IF, ID, EX, MEM, WB
  - ID和WB阶段都会用到寄存器堆，第2阶段读出是组合逻辑，第5阶段写入是时序逻辑
  - 数据存储器读出时可认为是组合逻辑，写入时是时序逻辑
- 在不同阶段之间增加流水线寄存器
  - 在WB阶段的寄存器堆可以被理解为第5级流水线寄存器
  - 命名法则：前级/后级
  - 寄存器开始，流经组合逻辑到寄存器结束
  - 时钟有效沿到来时，保存前级组合逻辑计算的结果，之后输出至下级组合逻辑
- 流水控制信号：与单周期相同的控制信号，流水传递



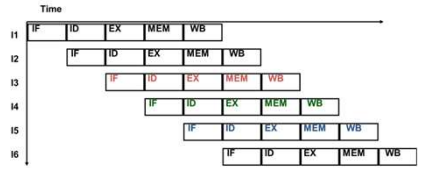
回顾：流水线的变化

- 在任何的时间片段，每一个阶段执行着*不同的*指令！
- 需要重新验证数据通路（连线和部件的放置）

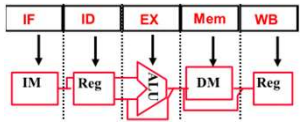


修正数据通路

回顾：流水线执行的表示



		IF级	ID/RF级	EX级	MEM级	WB级			
相对PC地址偏移	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	Instr 1	J 1	0→4	Instr 1	Instr 1				
4	Instr 2	J 2	4→8	Instr 2	Instr 2	Instr 1			
8	Instr 3	J 3	12→12	Instr 3	Instr 3	Instr 2	Instr 1		
12	Instr 4	J 4	12→16	Instr 4	Instr 4	Instr 3	Instr 2	Instr 1	
16	Instr 5	J 5	16→20	Instr 5	Instr 5	Instr 4	Instr 3	Instr 2	Instr1
20	Instr 6	J 6	20→24	Instr6	Instr6	Instr5	Instr4	Instr3	Instr2



回顾：流水线的性能

- 流水线允许使用相同的部件同时执行多条指令的某个部分
  - 指令级并行(ILP，*instruction level parallelism*)
  - 如果程序中相邻的一组指令是相互独立的，即不竞争同一个功能部件、不相互等待对方的运算结果、不访问同一个存储单元，那么它们就可以在处理器内部并行地执行
- 流水线的加速比

$$Number\ of\ stages \geq Speedup = \frac{Tc, single\_cycle}{Tc, pipelined}$$

  - 如果不均衡则加速比会下降
  - 加速比的获得是由于增加了吞吐量
    - 每条指令的延迟并没有减少

回顾：流水线冒险

1. *结构冒险*
   - 当处于两个流水段的指令需要同一个资源时会发生冲突
   - 解决方案 1：消除争用的起因
     - 复制资源或者提高资源的吞吐能力
   - 解决方案 2：检测资源争用，使其中一个争用流水段停顿
     - 让哪一个流水段停顿?

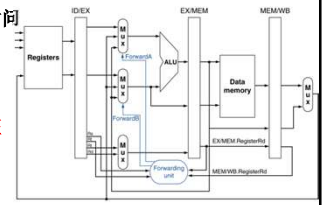
- 只使用一个memory的MIPS流水线
  - Load/Store需要访问内存, 取指令可能不得不暂停相应的周期
  - 流水线数据通路通常采用单独的指令和数据存储器

- MIPS流水线上寄存器堆访问存在的结构冒险，可能的解决方案:
  - 分割寄存器堆的访问周期: 时钟周期的前半段写，后半段读
  - 构建具有独立读/写端口的寄存器堆
  - 寄存器读/写可以在同一个时钟周期进行

回顾：流水线冒险

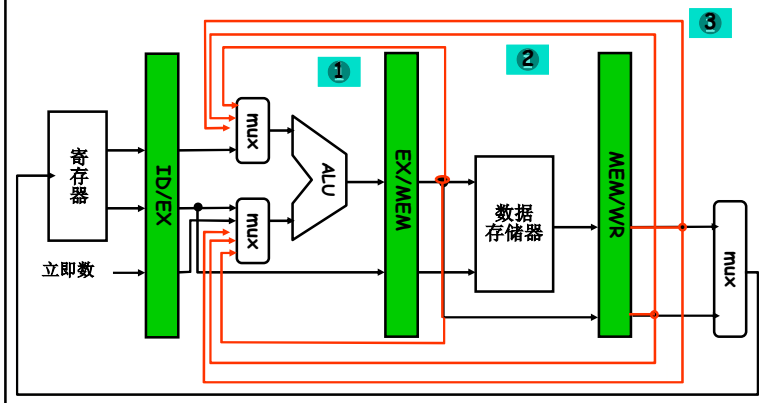
2. *数据冒险*
   - 指令之间的数据依赖
   - 需要等待之前的指令以完成数据读写

- 处理数据冒险
  - 读后写和写后写更容易处理
  - 五种处理写后读的基本方法
    - 检测并等待直到值在寄存器堆中可以访问
    - 检测并转发/旁路数据给相关的指令
    - 检测并消除相关性（在软件层面）
      - 不需要硬件检测相关性
    - 预测需要的值，“投机”执行，并且验证
    - 其它(细粒度多线程)
      - 不需要检测



- 互锁
  - 在流水线处理器中检测指令之间的相关性以确保执行正确
  - 基于软件 vs. 基于硬件

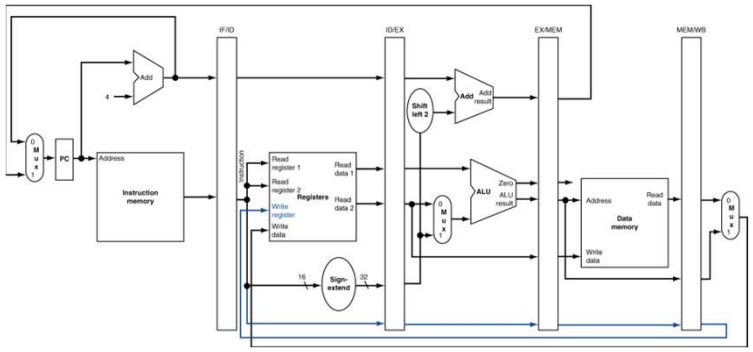
调整硬件结构支持旁路/转发



9

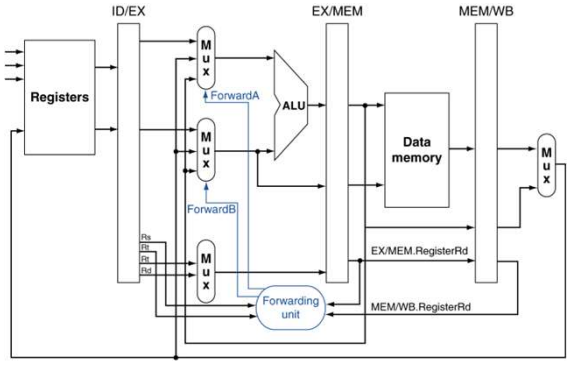
带旁路/转发的数据通路

- 需要改变什么？



10

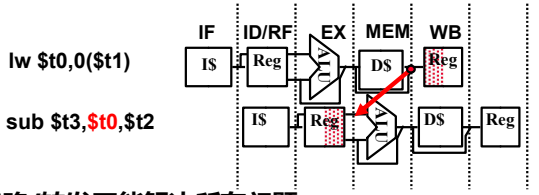
❖增加旁路/转发单元



11

数据冒险: Loads

❖数据流回到从前会带来风险



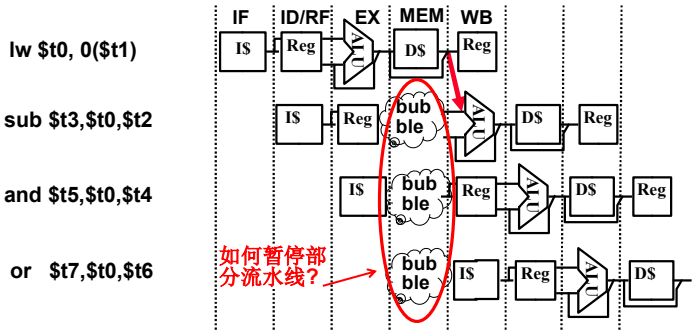
- 靠旁路/转发不能解决所有问题
  - 必须暂停依赖于load的指令，然后旁路/转发 (需要更多硬件)

12

❖硬件暂停流水线

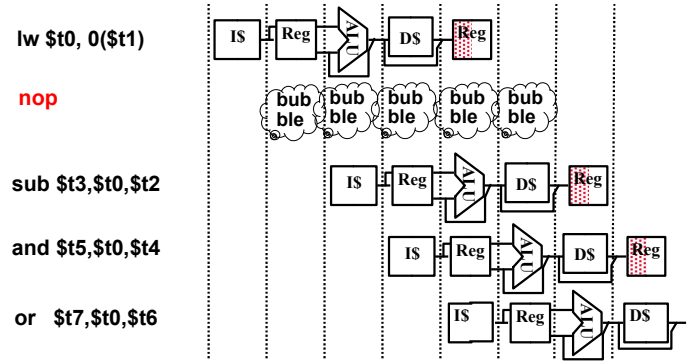
- “硬件互锁”

直观上看，这正是我们想要的，但是实际当中的暂停是“水平”实现的



13

❖暂停等价于nop



14

load导致的数据冒险：Clk0上升沿后

❖指令流

- lw进入IF流水段
- PC：PC + 4计算
- IM：输出lw指令到达IF/ID

15

load导致的数据冒险：Clk1上升沿后

❖指令流

- lw进入IF/ID, lw进入ID流水段，sub进入IF流水段
- PC：指向sub指令的地址
  - PC ← PC + 4
- IM：输出sub指令到达IF/ID

		IF级	ID级	EX级	MEM级	WB级	
地址 指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0 lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4 sub $t3, $t0, $t2								
8 and $t5, $t0, $t4								
12 or $t7, $t0, $t6								
16 add $t1, $t2, $t3								

16

load导致的数据冒险：Clk2上升沿后

- 指令流
  - Sub进入ID流水段，到达ID/EX寄存器；lw进入EX流水段，到达EX/MEM寄存器
- 冲突分析：冲突出现
- 执行动作：设置控制信号，在clk3插入nop指令
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC输出应保持为PC+4

			IF级	ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4								
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

load导致的数据冒险：Clk3上升沿后

- 指令流
  - Sub阻塞在IF/ID；lw进入MEM流水段，到达MEM/WB
  - ID/EX向ALU提供数据，由于控制信号清0，不影响后续指令
- 冲突分析：冲突解除
  - 转发机制将在clk4时可以发挥作用

			IF级	ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4	↑ 3	8→8	and	sub	nop	lw		
12	or $t7, $t0, $t6								
16	add $t1, $t2, $t3								

load导致的数据冒险：Clk4上升沿后

- 指令流
  - lw：结果写进MEM/WB, 进入WB流水段，到达RF。
  - sub：进入EX流水段，到达EX/MEM。ALU的操作数可以从WB段数据转发
- 执行动作
  - 控制MUX，使得WB段数据输入到ALU

			IF级	ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4	↑ 3	8→8	and	sub	nop	lw		
12	or $t7, $t0, $t6	↑ 4	8→12	and→or	and	sub	nop	lw结果	
16	add $t1, $t2, $t3								

load导致的数据冒险：Clk5上升沿后

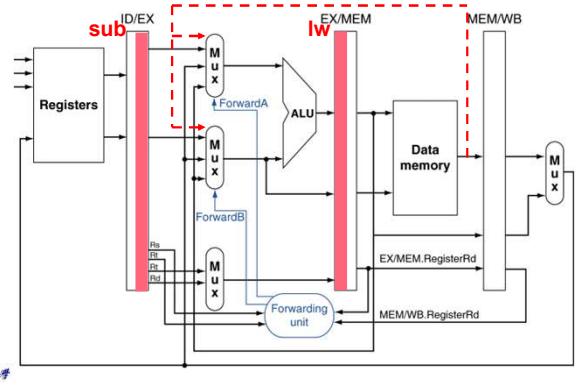
- 指令流
  - lw：结果写回至RF
  - sub：结果写进EX/MEM，进入MEM流水段

			IF级	ID级	EX级	MEM级	WB级		
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	↑ 1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	↑ 2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4	↑ 3	8→8	and	sub	nop	lw		
12	or $t7, $t0, $t6	↑ 4	8→12	and→or	and	sub	nop	lw结果	
16	add $t1, $t2, $t3	↑ 5	12→16	or→add	or	and	sub结果	nop	lw结果

load导致的数据冒险

❖Q：如果设置从DM到ALU输入的转发，这个设计优劣如何？

➢设计初衷：将DM读出数据提前1个clock转发至ALU，从而消除lw指令导致的数据相关，无需插入NOP



21

load导致的数据冒险

❖A：功能虽然正确，但CPU时钟频率大幅度降低

➢原设计： f = 5GHz

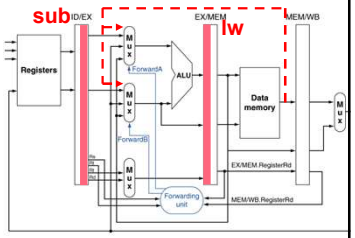
- 各阶段最大延迟为200ps

➢新设计： f = 2.5GHz

- EX阶段$_{修改后}$ = ALU延迟 + DM延迟 = 400ps
- EX阶段延迟成为最大延迟

警惕：木桶原理！

流水线各阶段延迟不均衡，将导致流水线性能严重下降



前面PPT的数据

Instr fetch	Register read	ALU op	Memory access	Register write
200ps	100 ps	200ps	200ps	100 ps

22

如何插入NOP指令?

❖检测条件：IF/ID的前序是lw指令，并且lw的rt寄存器与IF/ID的rs或rt相同

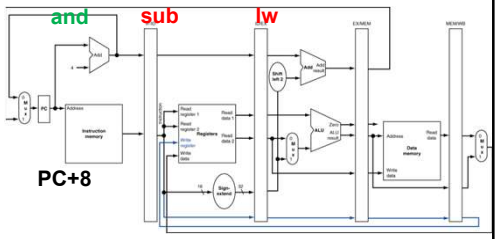
❖执行动作：

➢①冻结IF/ID：sub继续被保存

➢②清除ID/EX：指令全为0，等价于插入NOP

➢③禁止PC：防止PC继续计数，PC应保持PC+8

```
地址          指令
 0  lw   $t0, 0($t1)
 4  sub  $t3, $t0, $t2
 8  and  $t5, $t0, $t4
12  or   $t7, $t0, $t6
16  add  $t1, $t2, $t3
```

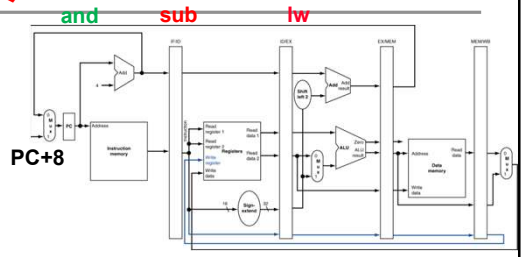


23

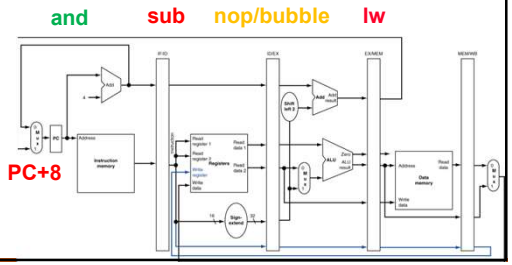
如何插入NOP指令?

Cycle N

```
地址          指令
 0  lw   $t0, 0($t1)
 4  sub  $t3, $t0, $t2
 8  and  $t5, $t0, $t4
12  or   $t7, $t0, $t6
16  add  $t1, $t2, $t3
```



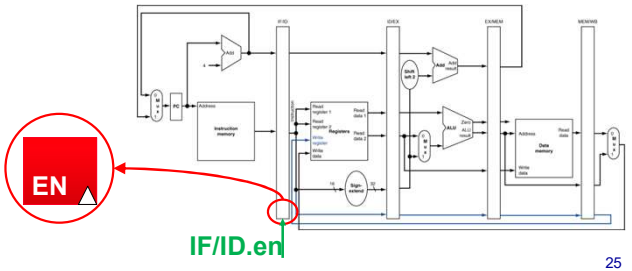
Cycle N+1



24

如何插入NOP指令?

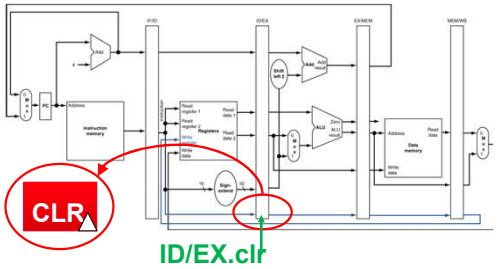
- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将IF/ID修改为使能型寄存器
- 控制系统：增加IF/ID.en控制信号
  - 当IF/ID.en为0时，IF/ID在下个clock上升沿到来时保持不变



25

如何插入NOP指令?

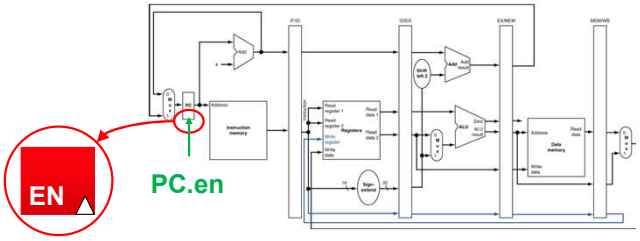
- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将ID/EX修改为复位型寄存器
- 控制系统：增加ID/EX.clr控制信号
  - 当ID/EX.clr为0时，ID/EX在下个clock上升沿到来时被清除为0



26

如何插入NOP指令?

- 执行动作：
  - ①冻结IF/ID：sub继续被保存
  - ②清除ID/EX：指令全为0，等价于插入NOP
  - ③禁止PC：防止PC继续计数，PC应保持为PC+4
- 数据通路：将PC修改为使能型寄存器
- 控制系统：增加PC.en控制信号
  - 当PC.en为0时，PC在下个clock上升沿到来时保持不变



27

如何插入NOP指令?

- lw冒险处理示例伪代码
- 注意：时序关系
  - 各信号在clk2上升沿后有效
  - NOP是在clk3上升沿后发生，即寄存器值在clk3上升沿到来时发生变化(或保持不变)

```
if (ID/EX.MemRead) &
   ((ID/EX.rt == IF/ID.rs) |
    (ID/EX.rt == IF/ID.rt))
    IF/ID.en ← 禁止
    ID/EX.clr ← 清除
    PC.en ← 禁止
```

地址	指令		IF级		ID级		EX级		MEM级		WB级	
		CLK	PC	IM	IF/ID		ID/EX		EX/MEM		MEM/WB	RF
0	lw $t0, 0($t1)	⤴ 1	0→4	lw→sub	lw							
4	sub $t3, $t0, $t2	⤴ 2	4→8	sub→and	sub		lw					
8	and $t5, $t0, $t4	⤴ 3	8→8	and	sub		nop		lw			
12	or $t7, $t0, $t6											
16	add $t1, $t2, $t3											

28

如果没有转发电路呢?

- 由于有转发电路，因此lw指令只插入1个NOP指令
- Q：如果没有转发，需要怎么处理呢？
- A：EX/MEM也需要做冲突分析及NOP处理
  - EX/MEM也需要修改，并增加相应控制信号

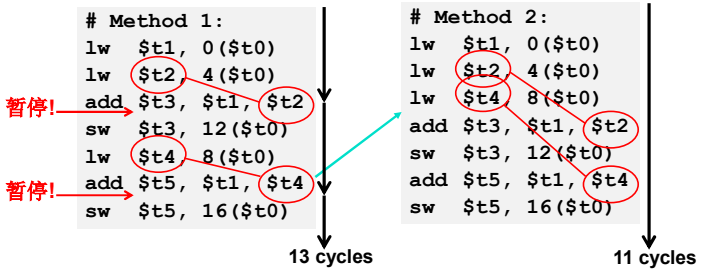
				IF级	ID级	EX级	MEM级	WB级
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	lw $t0, 0($t1)	1	0→4	lw→sub	lw				
4	sub $t3, $t0, $t2	2	4→8	sub→and	sub	lw			
8	and $t5, $t0, $t4	3	8	and	sub	nop	lw		
12	or $t7, $t0, $t6	4	8	and	sub	nop	nop	lw结果	
16	add $t1, $t2, $t3	5	8→12	and→or	and	sub	nop	nop	lw结果

load导致的数据冒险

- Load之后的时间片段称为load延迟时隙（slot）
  - 如果后续指令会用到load回来的结果，硬件互锁机制会暂停该指令一个时钟周期
  - 在延迟时隙由硬件暂停指令等价于在该时隙加nop (除非加nop占用更多的代码空间)
- Idea: 让编译器在该时隙插入一条不相关的指令 → 无需暂停!

避免暂停指令的代码调度

- 对代码重排序以避免下一条指令使用load的结果!
- MIPS 代码：A=B+E； C=B+F；



流水线冒险

3. 控制冒险

- 也叫控制相关或控制依赖，执行流依赖于之前的指令
- 数据冒险的特例：有关指令指针/程序计数器的数据相关

- 问题：下一个周期从PC里取出来的是什么?
- 答案：下一条指令的地址
  - 所有的指令都和他们之前的指令存在控制相关。为什么？
- 如果取到的指令不是一个控制指令:
  - 下一次取的PC是下一条顺序执行的指令
  - 只要我们知道取到的指令尺寸就行了
- 如果取到的指令是控制指令:
  - 我们如何决定下一个要取的PC?
- 实际上，我们怎么知道取的指令是不是一个控制指令?

分支的类型

类型	取指阶段能判断的分支方向	下一个可能地址的数量?	何时能够解析出下一个取指的地址?
条件分支	不知道	2	执行 (寄存器相关)
无条件分支	总是发生转跳	1	译码 (PC + offset)
调用	总是发生转跳	1	译码 (PC + offset)
返回	总是发生转跳	多	执行 (寄存器相关)
间接分支	总是发生转跳	多	执行 (寄存器相关)

不同类型的分支处理方式不同

如何处理控制冒险

- 关键在于使流水线保持充满正确的动态指令序列
- 当指令是控制指令时可能的解决方案有:
  - 停顿流水线直到得到下一条指令的取指地址
  - 猜测下一条指令的取指地址（分支预测）
  - 采用延迟分支（分支延迟槽/时隙）
  - 其它(细粒度多线程)
  - 消除控制指令（推断执行）
  - 从所有可能的方向取指（如果知道的话）（多路径执行）

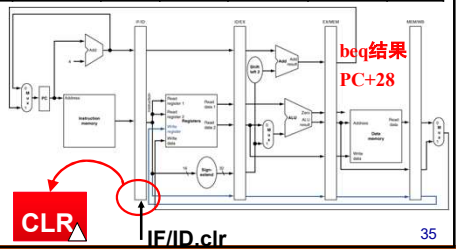
处理控制冒险：分支指令冒险造成的停顿代价

- 简单的解决方案：暂停每一个分支直到获得新的PC值
  - 我们必须暂停多长时间?

			IF级	ID级	EX级	MEM级	WB级	
地址	指令	CLK	PC	IM	IF/ID	ID/EX	EX/MEM	MEM/WB	RF
0	beq $1, $3, 6	1	0→4	beq→and	beq				
4	and $12, $2, $5	2	4	and	nop	beq			
8	or $13, $6, $2	3	4	and	nop	nop	beq结果		
12	add $14, $2, $2	4	4→28	and→lw	nop	nop	nop		
		5	28→32	lw→XX	lw	nop	nop	nop	nop
28	lw $4, 50($7)								

- 如不对分支指令做任何处理，则必须插入3个NOP
  - 分支指令结果及新PC值保存在EX/MEM，因此PC在clk4才能加载正确值
  - IF/ID在clk5才能存入转移后指令(即lw指令)

Q：IF/ID.clr表达式？



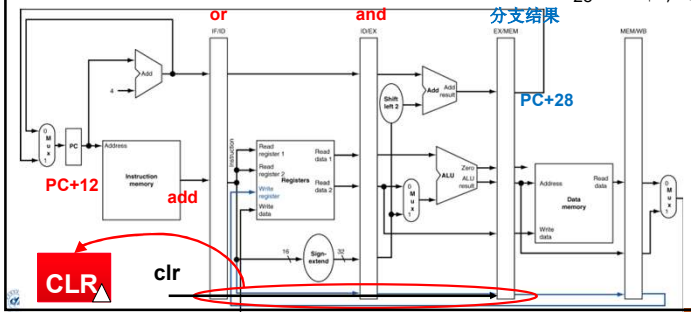
处理控制冒险: 分支

- *分支预测* – 猜测分支的结果，如果出错需要事后修正
  - 必须取消(*flush*)流水线中所有基于错误猜测执行的指令
  - 最终会取消多少指令?
- 如果预测所有的分支都不执行将得到最简单的硬件

方案1：假定分支不发生

- 即使在ID级发现是分支指令也不停顿
- 根据分支指令结果，决定是否清除3条后继指令
  - 使得and/or/add不能前进

PC相对偏移	指令
0	beq $1, $3, 6
4	and $12, $2, $5
8	or $13, $6, $2
12	add $14, $2, $2
28	lw $4, 50($7)



37

如何比停止取指更好…

- 与其等待关于PC的真相关被解决再行动，不如猜测下一个PC = PC+4，保持每个周期都取指
  这是好的猜测吗?
  如果猜错了会损失什么？
- ~20% 的指令组合是控制流
  - ~50 % 的 “向前” 控制流 (比如 if-then-else) 被执行
  - ~90% 的 “向后” 控制流 (比如 loop) 被执行
    总的来说，一般 ~70% 被执行， ~30% 不会执行
    [Lee and Smith, 1984]
- “下一个PC = PC+4” 的期望在~86% 的时间里是对的, 但是剩下那14%呢?

38

猜测下一个PC = PC + 4

- 总是预测下一条按顺序的指令就是下一条要被执行的指令
- 下一条取指地址和分支的预测方式
- 如何能让这种方式更有效率?
- 思路：使下一条按顺序的指令就是下一条要执行的指令的可能性最大
  - 软件：制定控制流图，使得“可能的下一条指令”出现在不发生分支的路径上
  - 硬件：???（如何能在硬件中做到这一点…）

39

猜测下一个PC = PC + 4

- 还能怎样使这种方式更高效?
- 思路：去掉控制流指令（或者尽量减少它的发生）
- 怎么做?
  1. 去掉不必要的控制流指令 → 组合推断(把条件推断组合起来)
  2. 将控制相关转化为数据相关 → 推断执行

40

性能分析

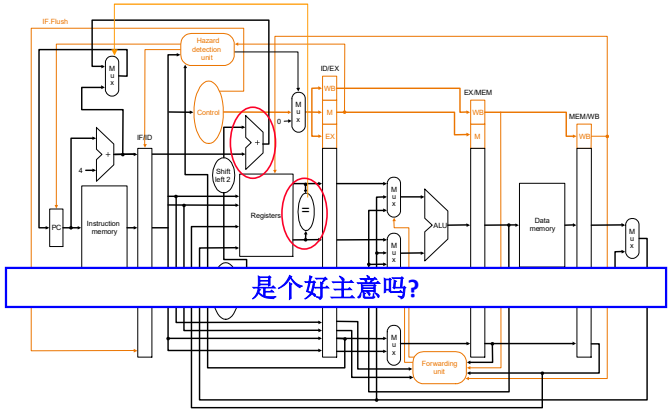
- 猜测正确 ⇒ 没有惩罚 ~86% 的时间
- 猜测不正确 ⇒ 2个气泡
- 假设
  - 没有数据相关
  - 20% 的控制流指令
  - 70% 的控制流指令发生转跳
  - CPI = [1 + (0.2*0.7) * 2]
    = [1 + 0.14 * 2] = 1.28

发生错误猜测的可能性　　错误猜测的惩罚

我们有可能减小这两者中的任何一个吗?

减小分支预测错误的代价

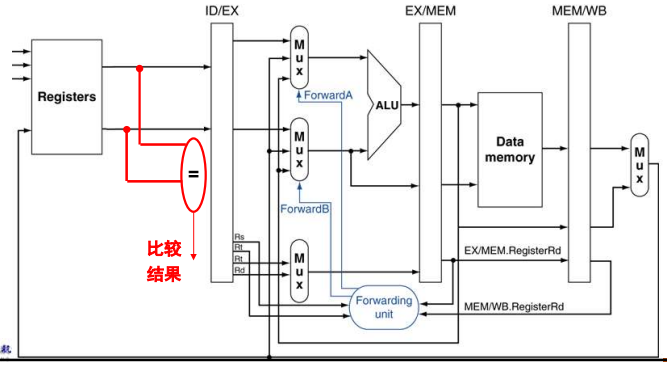
- 提前处理分支条件和获得目标地址（分支判断提前）



CPI = [1 + (0.2*0.7) * 1] = 1.14

方案2：缩短分支延迟

- 在ID阶段放置比较器，尽快得到分支指令结果
  - 分支指令结果可以提前2个clock得到
  - 分支指令后继可能被废弃的指令减少为1条
    - 当需要转移时，清除IF/ID即可



方案2：缩短分支延迟

- 比较器前置后，会产生数据相关
  - 分支指令可能依赖于前条指令的结果
- 依赖计算1：从ALU转发数据
- 依赖计算2：只能暂停

Q：如果依赖MEM/WB的结果，是否需要设置转发？

提示：MEM/WB已经有回写通道了，但RF设计满足吗？