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）

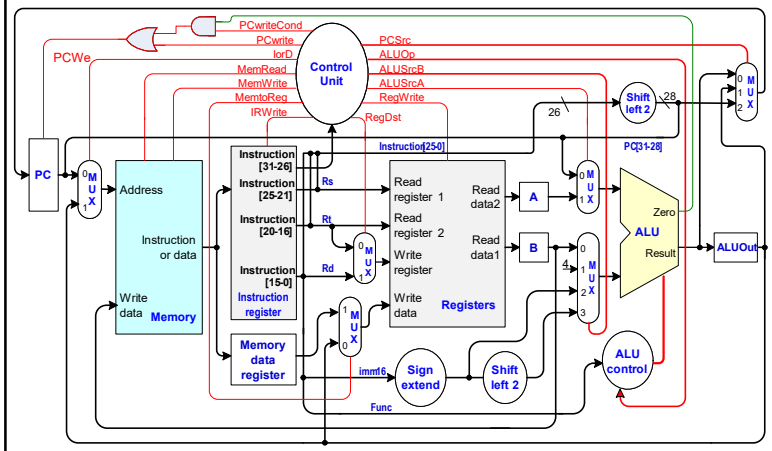
北京航空航天大学计算机学院中德所

栾钟治

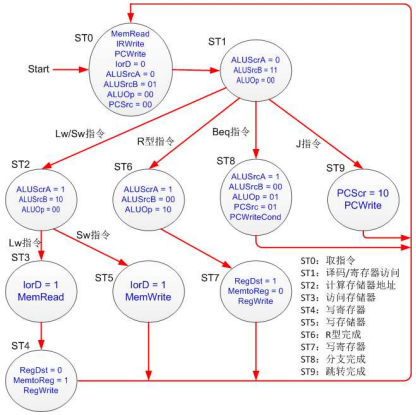
习题5——单周期处理器

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

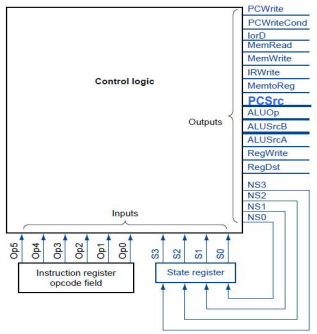
回顾：多周期完整数据通路与控制信号



回顾：多周期控制器主控单元



多周期主控单元有限状态图



- 有限状态机FSM
  - 状态编码用4位二进制表示
- 摩尔型FSM
  - 控制信号输出仅取决于当前状态
  - 次态由现态和当前输入决定

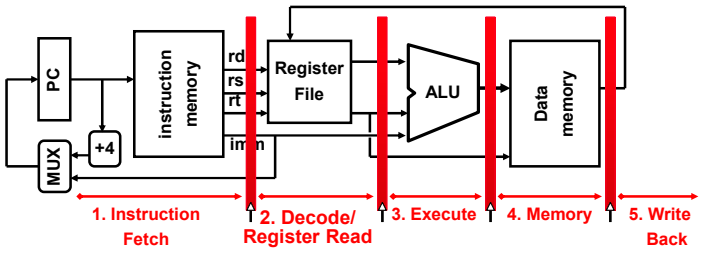
回顾：多周期性能分析

- 各型指令所需的时钟周期数和时间
  - R型指令：800ps
  - lw指令 ：1000ps
  - sw指令 ：800ps
  - beq指令 ：600ps
  - j指令 ：600ps
- 假设指令在程序中出现的频率
  - lw指令 ：25%
  - sw指令 ：10%
  - R型指令：45%
  - beq指令：15%
  - j指令 ：5%
- 则一条指令的平均CPI
  - 5*25%+4*10%+4*45%+3*15%+3*5% = 4.05
- 一条指令的平均执行时间：
  - 1000*25%+800*10%+800*45%+600*15%+600*5% = 810ps

回顾：计算系统的性能

- 响应时间与吞吐量
  - 响应时间：从提交作业到完成作业所花费的时间
  - 吞吐量：一定时间间隔内完成的作业数
- 单周期和多周期设计与响应时间的关系
- 流水线设计的性质
  - 流水线不改善单个任务处理延迟（响应时间），但改善了整体工作负载的吞吐
  - 流水线速率受限于最慢的流水段
  - 多个任务同时工作，但占用不同的资源
  - 潜在加速比 = 流水线级数
  - 流水段执行时间不平衡，则加速比下降
  - 填充流水线和排放流水线，加速比下降

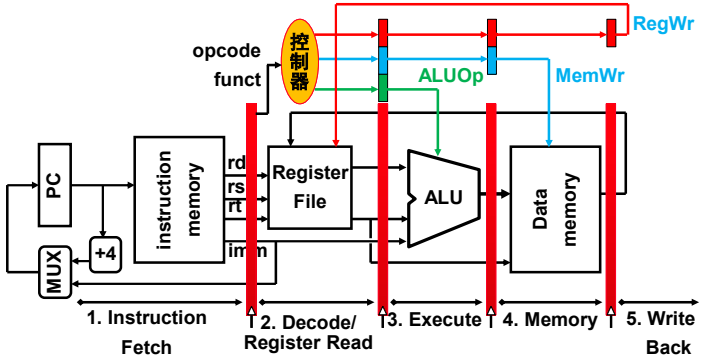
流水线数据通路



- 在不同阶段之间增加寄存器
  - 保存前一个周期产生的信息
- 5 阶段流水线
  - 意味着时钟频率实际上可以看作变为原来的5倍

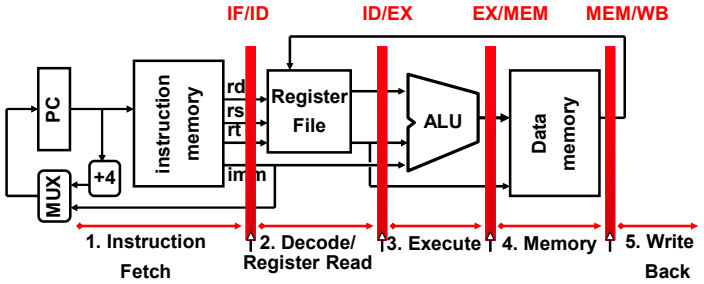
流水的控制信号

- 控制器：译码产生控制信号，与单周期完全相同
- 控制信号流水寄存器：控制信号在寄存器中传递，直至不再需要



正确认识流水线—流水线寄存器

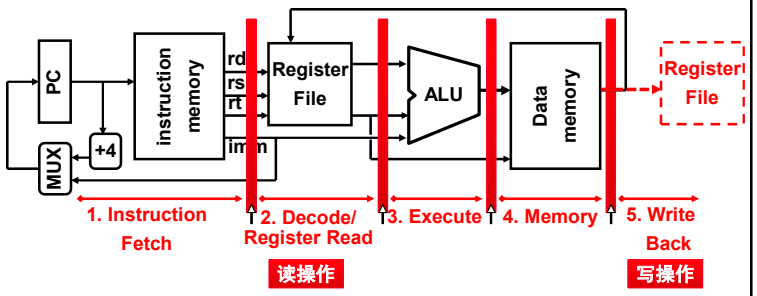
- 命名法则：前级/后级
  - 示例：IF/ID，前级为读取指令，后级为指令译码(及读操作数)
- 功能：时钟有效（比如上升）沿到来时，保存前级结果；之后输出至下级组合逻辑
  - 也可能直接连接到下级流水线寄存器
    - 例如ID/EX保存的从RF读出的第2个寄存器值，就直接传递到EX/MEM



9

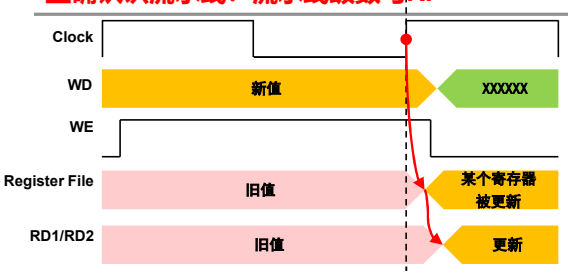
正确认识流水线—流水线级数与RF

- N级流水线：必须有N级流水线寄存器
  - 插入N-1级流水线寄存器，最后一级为Register File
- RF：这是一个特殊部件，有2次使用
  - 读：第2级；写：第5级



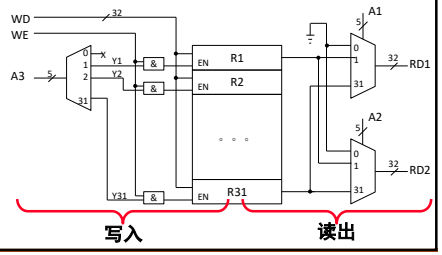
10

正确认识流水线：流水线级数与RF



- RF具有2重行为
  - 读出：组合逻辑
  - 写入：时序逻辑(寄存器)

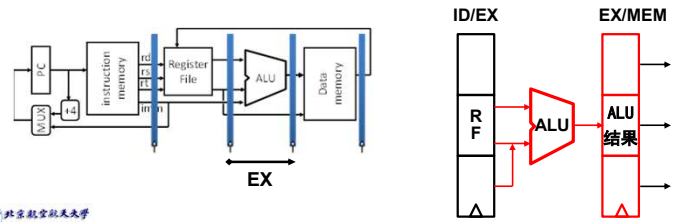
- RF：上升沿被更新
  - WE有效，则WD被写入A3
- 输出：与时钟无关，仅与RF内容及A1/A2相关



11

正确认识流水线：流水阶段与流水线寄存器

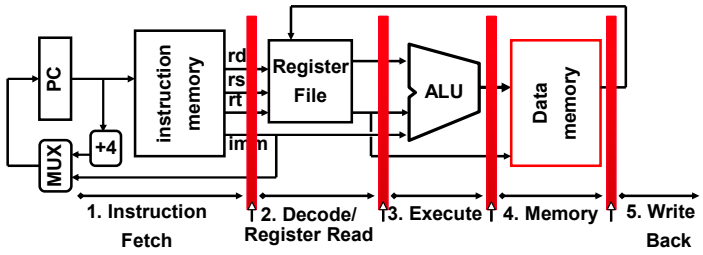
- 流水阶段：组合逻辑+寄存器
  - 起始：前级流水线寄存器的输出
  - 中间：组合逻辑（如ALU）
  - 结束：写入后级流水线寄存器
  - 当时钟上升沿到来时，组合逻辑计算结果存入后级寄存器
- 示例：EX阶段
  - 起始：ID/EX流水线寄存器中的RF寄存器/扩展单元的输出
  - 中间(组合逻辑)：ALU完成计算
  - 结束(寄存器)：在clock上升沿到来时，结果写入EX/MEM中相应寄存器



12

正确认识流水线：DM

- 写入时：表现为寄存器，属于MEM/WB寄存器范畴
- 读出时：可以等价为组合逻辑
  - 与RF的读出是类似的



13

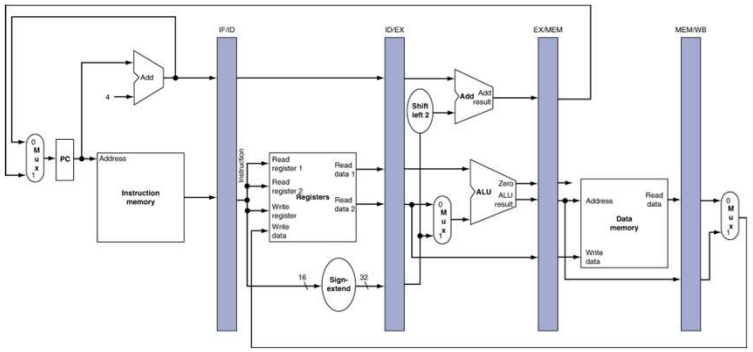
流水线的变化

- 寄存器影响信息的流动
  - 寄存器命名 (例如：IF/ID，标明相邻的阶段)
  - 寄存器*分隔*各阶段之间的信息流
  - 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）

14

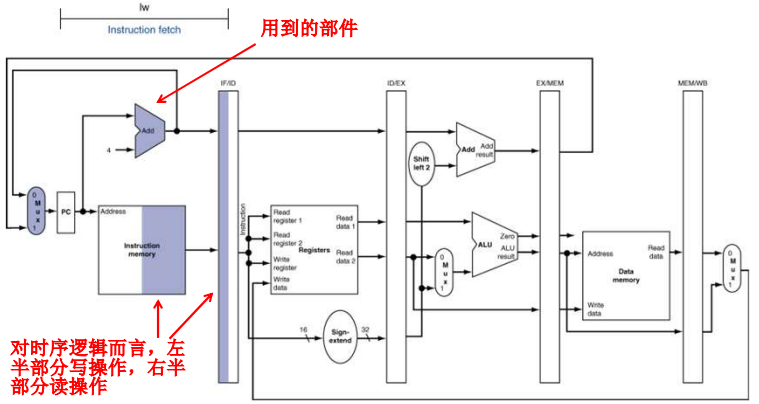
流水线的细节分析

- 验证lw指令在流水线中的流动



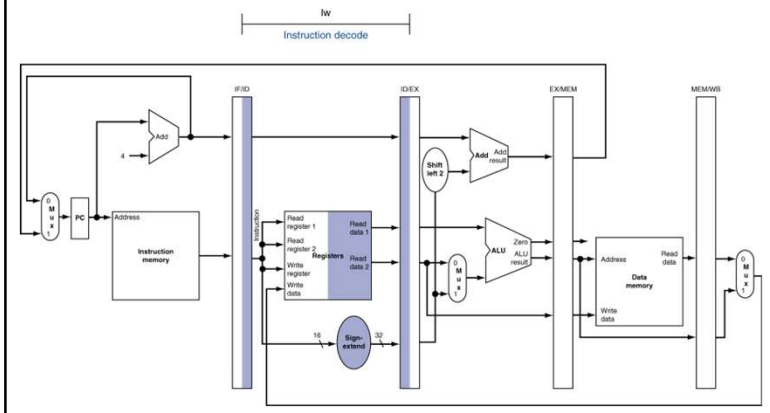
15

Lw指令的取值（IF）



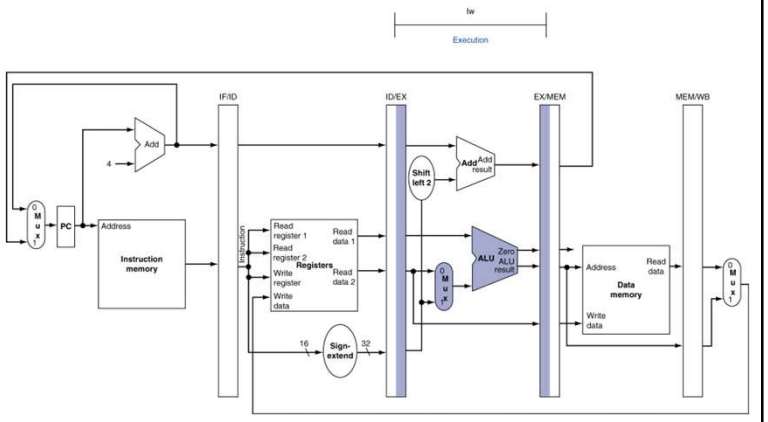
16

Lw指令的译码（ID）



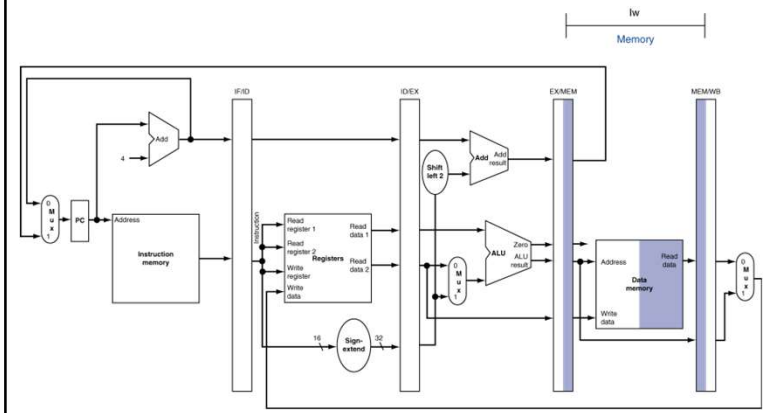
17

Lw指令的执行（EX）



18

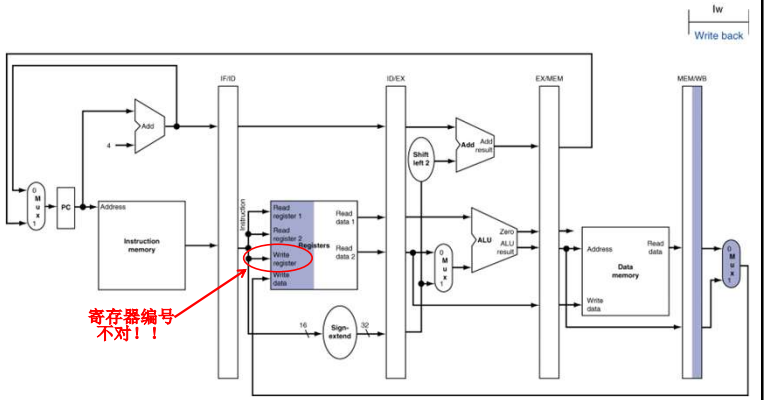
Lw指令的访存（MEM）



19

Lw指令的写回（WB）

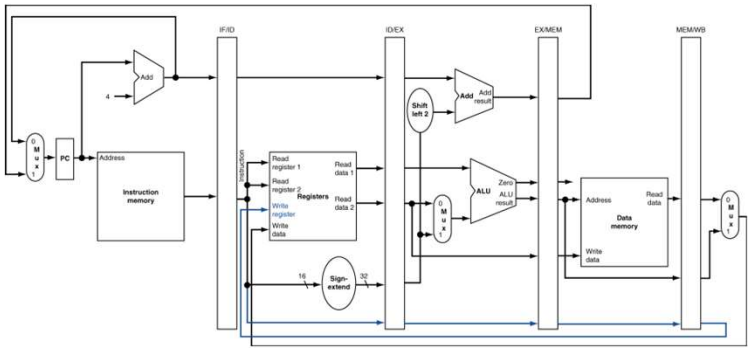
这里有些不对头！！



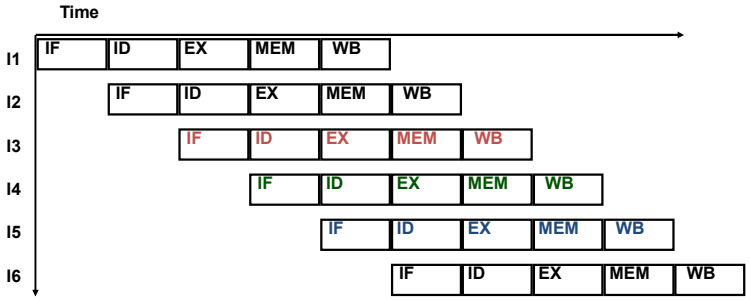
20

修正数据通路

- 这样，任何指令写寄存器就不会有问题了！



流水线执行的表示



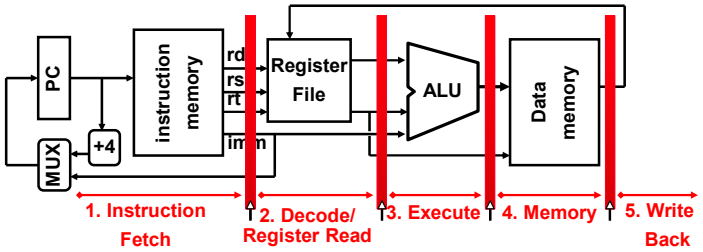
- 所有的指令必须有同样的阶段数，所以有些指令在某些阶段会处于空闲状态（idle）
  - 例如MEM阶段对所有算术运算指令

时钟驱动的流水线时空图

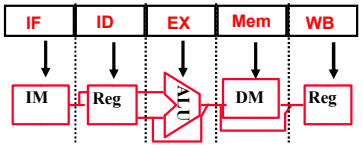
- 用途：精确分析指令/时间/流水线3者关系时
  - 行：某个时钟，指令流分别处于哪些阶段
  - 列：某个部件，在时间方向上执行了哪些指令
- 注意区分流水阶段与流水线寄存器的关系
- 可以看出，在clk5后，流水线全部充满
  - 所有部件都在执行指令
    - 只是不同的指令

				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	↑ 1	0→4	Instr 1	Instr 1				
4	Instr 2	↑ 2	4→8	Instr 2	Instr 2	Instr 1			
8	Instr 3	↑ 3	12→12	Instr 3	Instr 3	Instr 2	Instr 1		
12	Instr 4	↑ 4	12→16	Instr 4	Instr 4	Instr 3	Instr 2	Instr 1	
16	Instr 5	↑ 5	16→20	Instr 5	Instr 5	Instr 4	Instr 3	Instr 2	Instr1
20	Instr 6	↑ 6	20→24	Instr6	Instr6	Instr5	Instr4	Instr3	Instr2

流水线图

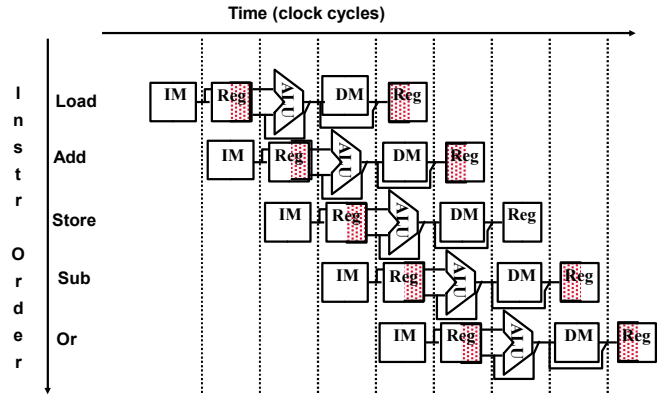


- 使用数据通路图表达流水线



流水线的图形化表示

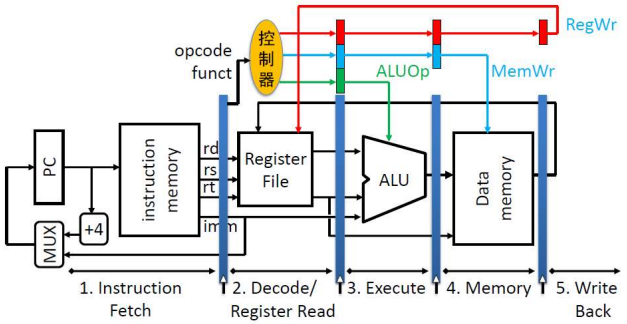
- 寄存器堆: 右半部读, 左半部写



25

指令级并行(ILP)

- 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
  - *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



26

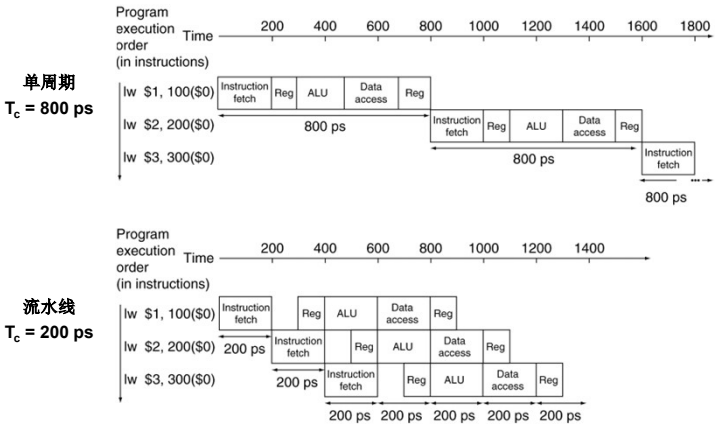
流水线的性能

- 假设每个阶段的时间如下
  - 寄存器读/写100ps
  - 其他阶段200ps

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- 流水线的时钟频率?
  - 对比流水线数据通路和单周期数据通路

27



28

流水线的加速比

- 使用T_c (完成一条指令的平均时间) 计算加速比

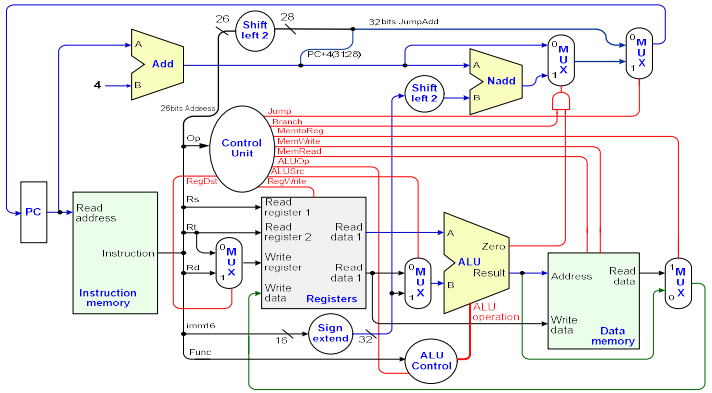
$$T_{c,pipelined} \geq \frac{T_{c,single-cycle}}{Number\ of\ stages}$$

  - 各阶段均衡的时候取等号 (各阶段消耗相等的时间)
- 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

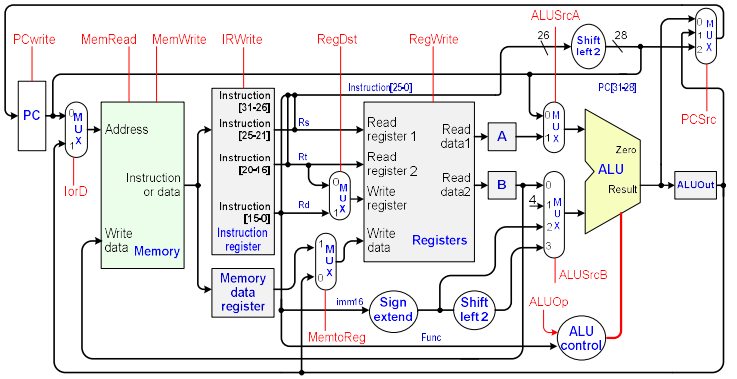
流水线和ISA设计

- MIPS的指令集就是为流水线设计的！
- 所有的指令都是32-bits
  - 方便在一个周期内完成取指和译码
- 指令格式简单规范, 2个源操作数域的位置固定不变
  - 能够在一步内译码和读寄存器
- 只有Load和Store指令操作Memory
  - 能够在第三阶段计算地址，第四阶段访存
- 内存对齐
  - 访存只需一个周期

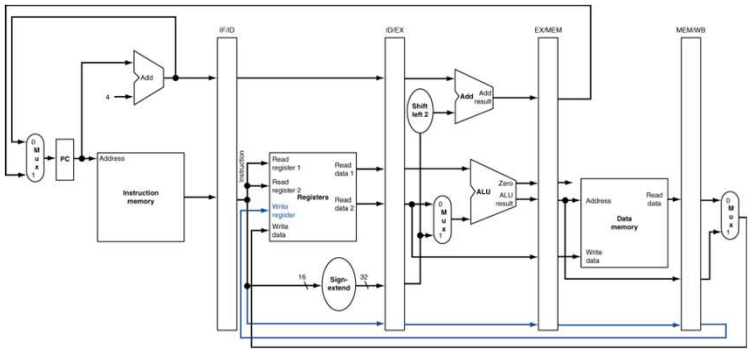
单周期数据通路



多周期数据通路



❖流水线数据通路



流水线冒险

❖也叫“依赖” 或者“相关”

- 两种与程序语义相关的基本类型，表明了指令之间关于“序”的需求

在下一个时钟周期妨碍下一条指令执行的状况

1) *结构冒险*
   - 也叫资源相关或者冒险，不是由程序语义表明的类型
   - 某个需要的资源忙 (比如在多个阶段都要用到)
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
3) *控制冒险*
   - 执行流依赖于之前的指令

处理结构冒险

❖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
  - 例如，将指令存储器（Cache）和数据存储器（Cache）分开
  - 例如，为存储结构设计多个端口

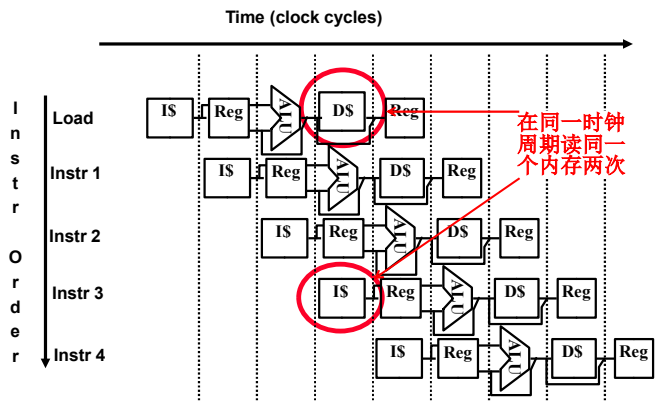
❖解决方案 2：检测资源争用，使其中一个争用流水段停顿
- 让哪一个流水段停顿?
- 例如：如果你的寄存器堆只有一个读写端口会怎么样?

处理结构冒险

❖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
  - 产生一个流水线“气泡”

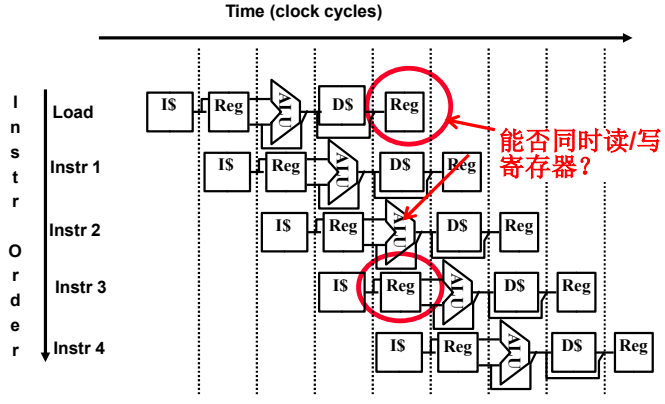
❖因此,流水线数据通路需要单独的指令和数据存储器
- 单独的 L1 I$ 和 L1D$

结构冒险#1: 存储器



37

结构冒险#2: 寄存器堆



38

处理结构冒险

- 两种可能的解决方案:
  1) 分割寄存器堆的访问周期: 时钟周期的前半段写，后半段读
     - 可行，因为寄存器堆的访问速度非常快 (小于ALU阶段所需时间的一半)
  2) 构建具有独立读/写端口的寄存器堆
- 结论: 寄存器读/写可以在同一个时钟周期进行

39

数据冒险的种类

指令之间的数据依赖

$r_3 \leftarrow r_1$ op r_2 $r_5 \leftarrow r_3$ op r_4	写后读 (RAW)	又叫流相关，关于值的相关性，是真相关
$r_3 \leftarrow r_1$ op r_2 $r_1 \leftarrow r_4$ op r_5	读后写 (WAR)	又叫反相关，关于名字的相关性，是伪相关
$r_3 \leftarrow r_1$ op r_2 $r_5 \leftarrow r_3$ op r_4 $r_3 \leftarrow r_6$ op r_7	写后写 (WAW)	又叫输出相关，关于名字的相关性，是伪相关

40

如何处理数据冒险

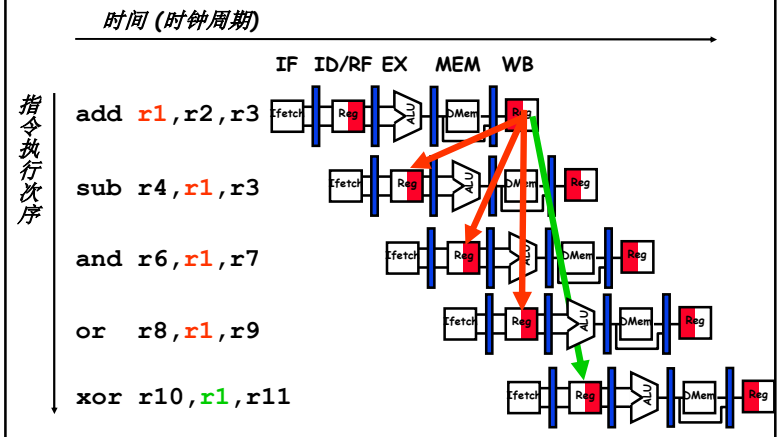
- 读后写和写后写更容易处理
  - 在一个阶段中完成写操作并且保证程序序
- 五种处理写后读的基本方法
  - 检测并等待直到值在寄存器堆中可以访问
  - 检测并转发/旁路数据给相关的指令
  - 检测并消除相关性（在软件层面）
    - 不需要硬件检测相关性
  - 预测需要的值，“投机”执行，并且验证
  - 其它（细粒度多线程）
    - 不需要检测

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
- 基于软件的互锁

  vs.
- 基于硬件的互锁
- MIPS的首字母缩写?
  - Microprocessor without Interlocked Piped Stages

数据冒险



数据冒险解决策略 - 旁路/转发