## 实验十一气泡流水线设计实验

### 实验目的

1. 掌握各种流水线冲突的概念。
2. 掌握插入气泡解决各种冲突的原理。
3. 学习判断流水线冲突的逻辑单元的设计方法。

### 实验原理与实验内容

#### 流水线冲突的基本概念与流水线的控制方法

##### 流水线冲突的基本概念

理想流水线所有待加工对象均需要通过相同的流水级，不同流水级之间无共享资源，且各流水级运行时长一致，进入某一流水级的对象也不会受到其他流水级对象的影响，但这仅仅适合工业生产流水线。计算机指令流水线存在较多的指令相关，会引起流水线的冲突和停顿。

指令相关是指在指令流水线中，某条指令的某个阶段必须等待前面某条指令的某个阶段完成才能开始。即这两条指令之间存在某种依赖关系，称它们之间存在指令相关。指令相关包括数据相关、结构相关和控制相关。指令相关会导致流水线出现冲突/冒险（Hazard）。流水线冲突是指由于指令相关的存在，导致指令流水线出现“断流”或“阻塞”，下一条指令不能在预期的时钟周期加载到流水线中。流水线冲突包括数据冲突、结构冲突、控制冲突。

①数据冲突

当一条指令需要访问的数据还没有被前面另一条指令计算出来时，就会发生数据冲突。根据指令读访问和写访问的顺序进行排列，常见有3种数据冲突：写后读（Read after Write，RAW）冲突、读后写（Write after Read，WAR）冲突、写后写（Write after Write，WAW）冲突。



数据冲突

a）写后读冲突。如果一条指令的源操作数是前面另一条指令的目的操作数，这种数据冲突就称为写后读（RAW）冲突。图7-1中的I1和I2、I1和I3间都存在写后读冲突：I2的x3是源操作数，值来自I1的目的操作数写入结果；I3的x3是源操作数，值来自I1的目的操作数写入结果。I2在第3个时钟周期到达译码级开始读取源操作数；I3在第4个时钟周期到达译码级开始读取源操作数。对于I1而言，该指令需要在第5个时钟周期才能运行到写回级将目的操作数的结果写回寄存器堆完成x3的更新，因此I2和I3均不能读到正确的x3值，会导致运行出错。

b）读后写冲突。如果一条指令的目的操作数是前面另一条指令的源操作数，这种数据冲突就称为读后写（WAR）冲突。图7-1中I2和I3之间存在读后写冲突：I2的源操作数x4是I3的目的操作数。在第7个时钟周期，指令I3运行到写回级对x4完成了更新，而I2在第3个时钟周期已经读取过x4的值，所以这种数据相关对**顺序流水线**指令的执行没有任何影响。

c）写后写冲突。如果一条指令的目的操作数和前面另一条指令的目的操作数是相同的，这种数据冲突就称为写后写（WAW）冲突。图7-1种I3和I4之间存在写后写冲突：I4的目的操作数x4也是I3的目的操作数。I3在第7个时钟周期达到写回级对x4完成更新，I4在第8个时钟周期对x4完成更新，最终x4的取值为I4的值，与预期一致，因此这种数据相关对顺序流水线指令的执行没有影响。

②结构冲突

当多条指令在同一时钟周期都需要使用同一个功能单元而引起的冲突称为结构冲突。对于单发射流水线而言，假如流水线仅有一个存储器（冯诺依曼结构），数据和指令都存放在同一个存储器中，图7-2中I1为存数指令，在第3个时钟周期进入执行级访问存储器进行存数，而此时正在发送I4的地址至存储器，这时就会发生访存结构冲突。访存结构冲突实际上在单周期处理器中就存在，一种解决方法是采用独立的指令存储器和数据存储器，目前SRAM结构的MyCPU就对应了这种解决办法——在第3个时钟周期，I1访问DataMEM，I3访问InstMEM，两者互不干扰。



结构冲突

③控制冲突

当流水线遇到转移指令或者其他会改变PC值的指令时，转移指令后续已经载入流水线的相邻指令可能不能进入执行阶段，这种冲突称为控制冲突。对于转移指令中的条件分支而言，指令是否发生跳转以及跳转的目标地址需要指令运行至执行级中通过BRU进行判断，而条件分支指令相邻的后续若干指令已经预取进入流水线（后续预取指令条数称为预取深度）。当条件分支指令判断为发生跳转时，流水线预取的指令不应进入流水线执行，此时需要清空预取指令，使得条件分支指令的下一条指令为正确的跳转目标地址的指令。如图7-3所示，在第3个时钟周期时，假如I1被判断为发生跳转，则I1的下一条指令应该为InstMEM中地址80002000对应的指令，而此时I2和I3已经进入了流水线，这2条指令应当被冲刷清空。发生控制冲突时，流水线会清空预取指令，浪费了若干时钟周期，会引起流水线性能降低。



控制冲突

##### 流水线的控制方法

流水线分为集中式控制和分布式控制。对于五级流水线而言，所以的数据均存于各级缓存中，只需要对缓存内的数据进行合理控制便能完成对流水线的控制。

①分布式控制

在分布式控制中，流水线的控制逻辑被分散在各个流水级（取指单元、译码单元……）中，每个流水级都有自己的控制逻辑来管理其执行的指令或操作。这种方式下，每个流水级都可以独立地对指令进行控制和处理，因此具有一定的并行性和灵活性。分布式控制可以提高流水线的吞吐量，并且通常用于高性能的超标量处理器设计中。

如图7-4所示，每两级缓存间互相发送valid和ready信号进行流水线的控制，valid表示当前data数据包是否有效，ready表示下一级能否接受当前data数据包。valid信号随data信号向下流动，而ready信号类似线网类型，并不会保存在寄存器中。当下一级可以接受当前数据包时，当前级发送给上一级的ready才能为1，且缓存更新data内容。若下一级的ready信号为0，则当前缓存继续保持原data信号不变且向上一级发送ready=0。



分布式控制结构

如图7-5所示，在T1时刻假设指令I3在执行单元遇到了某些情况需要请求暂停，此时执行级缓存接收到执行单元的暂停请求（简化为图7-5中MemoryStage发送的ready=0）并向译码级缓存发送ready=0，译码级缓存接收到执行级的ready信号也会向前发送ready=0信号。由于下一级的ready不为1时，执行级缓存、译码级缓存和取指级缓存都会继续保存data值不发生更新，同时执行单元会向下一级发送valid=0信号，表示当前的data包无效。缓存级内data停止更新导致指令不会向后发生流动，因此在T2时刻I5、I4、I3仍在原级停留，而T1时刻访存级收到的ready=1，因此指令会在T2时刻进入写回级（写回级为最后一级，所以其收到的ready信号恒为1）。在T3时刻，执行级解除暂停（简化为图7-5中MemoryStage发送的ready=1），所有执行级及前面流水级的指令可以继续向下流动。各级之间的ready和valid逻辑均在各级内部产生，因此为分布式控制。



分布式控制信号和指令时空图

②集中式控制

在集中式控制中，整个流水线的控制逻辑由一个中央控制单元或者类似的单个控制逻辑单元负责。这个单元负责协调和管理整个流水线的指令流动，包括指令的执行、流水线的暂停等操作。集中式控制通常更容易实现和调试，但可能会成为流水线的瓶颈，限制了流水线的并行性和性能。

图7-6展示了集中式控制下的五级流水线结构，所有的ready和valid信号均发送给一个中央控制单元，所有的控制逻辑由控制单元产生。



集中式控制结构

在实际的处理器设计中，通常会根据具体的需求和性能目标选择适合的控制方式。有些处理器可能会采用分布式控制和集中式控制相结合的方式，以平衡性能和复杂性之间的关系。

#### 流水线冲突的判断与气泡解决方法

##### 气泡解决方法

如图7-5所示，T1时刻执行级发生暂停，执行级之前的流水级均发生停顿，但之后的流水级依旧可以运行，其实现方法就是不断的向后面的流水级插入空操作，这种空操作的插入在流水线中类似气泡的产生，因此被称为气泡解决办法。在T2时刻的访存级中并无指令就如同塞入了一个气泡。

##### 数据冲突

对于单发射顺序五级流水线MyCPU而言，仅存在写后读冲突，只需对该冲突进行解决即可。一条指令的源操作数是前面另一条指令的目的操作数时才会发生写后读冲突。当指令处于译码单元时访问寄存器堆读取源操作数，指令处于写回单元时访问寄存器堆更新目的操作数。那么只需要使得在译码单元的指令暂停，等待前面的指令运行至写回单元完成对其需要的源操作数的更新后再继续运行译码单元的指令即可。对于五级流水线而言，译码级后还有3级：执行级、访存级、写回级，只需对这3级中的指令进行检查，查看其目的操作数是否与译码级中的某个源操作数一致即可，如果一致且目的操作数需要写回，则将译码级中的指令暂停，否则译码级中的指令就无需暂停。

##### 结构冲突

MyCPU目前采用SRAM接口，有InstMEM和DataMEM两个存储器，因此不存在结构冲突。

##### 控制冲突

MyCPU的转移指令在执行级处理。发生转移时需要清空取指级和译码级，这种操作也被称为flush。下面有两种处理办法：

①将取指单元和译码单元的valid置为0

这样进入缓存时的数据被标记为无效，下一级只会执行空操作。



修改处理单元的valid

②清除译码级缓存和执行级缓存

跳转时直接清空缓存级的数据，类似于reset，这样子下一级收到的数据也是无效，也会执行空操作。



清空缓存级的数据

#### 气泡流水线的数据通路的控制设计

集中式控制方法只需给每个部件增加一个和中央控制单元的交互的接口即可，之后只需要修改控制单元内部的控制逻辑即可，因此更容易实现。MyCPU的控制逻辑将以集中式控制为主。

在流水线中增加一个控制单元Ctrl，流水线的整体结构如图7-6所示。Ctrl接收译码单元、执行单元、访存单元、写回单元这4个处理单元的info信号，用于数据冲突的判断。Ctrl产生并发送allow\_to\_go和do\_flush两个信号，分别用于控制流水线缓存级的更新和缓存级的清空。在信号名前加上流水级名称表示信号的发送目标，如decodeStage.allow\_to\_go=1表示译码级缓存可以更新为取指单元发送来的data数据包；decodeStage.do\_flush=1表示译码级缓存清空寄存器值。

在满足下面任一条件时流水线发生数据冲突decodeStage.allow\_to\_go为0：

①executeUnit.info.reg\_wen && executeUnit.info.reg\_waddr.orR[[1]](#footnote-1) && decodeUnit.info.src1\_ren && decodeUnit.info.src1\_raddr == executeUnit.info.reg\_waddr

②memoryUnit.info.reg\_wen && memoryUnit.info.reg\_waddr.orR && decodeUnit.info.src1\_ren && decodeUnit.info.src1\_raddr == memoryUnit.info.reg\_waddr

③writeBackUnit.info.reg\_wen && writeBackUnit.info.reg\_waddr.orR && decodeUnit.info.src1\_ren && decodeUnit.info.src1\_raddr == writeBackUnit.info.reg\_waddr

fetchUnit.allow\_to\_go和decodeStage.allow\_to\_go一致，因为当译码级阻塞时，取指级也应当阻塞。

执行级、访存级和写回级无阻塞需求，因此executeUnit.allow\_to\_go、memoryUnit.allow\_to\_go和writeBackStage.allow\_to\_go恒为1。

执行单元发生跳转时需要清空前面几级缓存的数据，因此decodeUnit.do\_flush和executeUnit.do\_flush的值都等于executeUnit.branch。因为取指单元并不存在所谓的取指级缓存，因此fetchUnit.do\_flush恒为0。发生转移时后续缓存级无需清除，因此memoryUnit.do\_flush和writeBack.do\_flush值恒为0。至此完成了Ctrl生成的所有信号的定义。

#### 气泡流水线的数据通路的前端设计

##### 取指单元

前端需要解决发生数据冲突时的流水线暂停以及发生控制冲突时的流水线冲刷。

当译码单元发生暂停时，取指单元也需要暂停，只需将PC停止更新即可完成取指级的暂停。如图7-9所示，当Ctrl发送来的fetchUnit.allow\_to\_go为1时，第一个数据选择器会选择PC+4使得PC能顺序更新，反之选择PC使得PC保持不变达到暂停效果。



取指单元结构

##### 译码级缓存

译码级缓存结构如图7-10所示。当decodeUnit.allow\_to\_go为1时寄存器值更新为data1，否则值保持不变。将各级缓存中寄存器清零信号定义为clear，当clear为1时清零寄存器。当decodeUnit.do\_flush为1时，寄存器值全部清零，因此译码级缓存的clear等于decodeUnit.do\_flush。



译码级缓存结构

#### 气泡流水线的数据通路的后端设计

##### 译码单元

译码单元需要将译码得到的info信号发送给控制单元Ctrl，以便在控制单元中进行冲突检测。译码单元结构如图7-11所示。



译码单元结构

##### 执行级缓存和执行单元

执行级缓存修改后的结构与译码级缓存类似，但是其也需要decodeUnit.allow\_to\_go信号。执行级缓存结构如图7-12所示。寄存器值的更新取决于executeUnit.allow\_to\_go。clear的定义有所不同，当满足下面任一情况时，执行级缓存清零，clear为1：

①executeUnit.do\_flush等于1时，需要清空寄存器。

②executeUnit.allow\_to\_go等于1而decodeUnit.allow\_to\_go等于0，即发生了数据冲突，译码单元需要暂停，这时也需要清空寄存器，相当于插入一个气泡。

执行单元需要发送info和branch信号给控制单元。执行单元结构如图7-13所示。



执行级缓存结构



执行单元结构

##### 访存级缓存和访存单元

访存级缓存的修改和译码级缓存的修改相同，寄存器值的更新取决于memoryUnit.allow\_to\_go，clear信号的值等于memoryUnit.do\_flush。

访存单元的修改和译码单元相同，只需发送info信号给控制单元。

##### 写回级缓存和访存单元

写回级缓存的修改和访存级缓存的修改相同，寄存器值的更新取决于writeBackUnit.allow\_to\_go，clear信号的值等于writeBackUnit.do\_flush。

写回单元的修改和访存单元的修改相同，只需发送info信号给控制单元。

### 实验要求

1. 在上一个实验的基础上将理想流水线MyCPU修改为气泡流水线。
2. 通过本实验提供的所有仿真验证测试用例。
3. 通过本实验提供的所有板级验证测试用例。
4. 线上提交实验作品，获得系统评分。
5. 撰写实验报告。报告中应包含以下内容和至少3道本实验思考题。
6. 仿照图7-5，绘制集中式控制信号和指令时空图。
7. 修改MyCPU内部数据通路图，将理想流水线升级为气泡流水线。

### 实验步骤

1. 实验步骤参见上一实验的实验步骤部分。

### 思考与探索

1. 读后读（RAR）冲突属于数据冲突吗？为什么？尝试结合图7-1分析一下。
2. 读后写冲突与写后写冲突对于顺序流水线无影响，其对于乱序流水线有影响吗？又该如何解决？尝试举例说明。
3. 对于超标量流水线而言，还可能存在什么样的结构冲突？对于这些冲突又可以如何解决？尝试举例说明。
4. 如何减轻由于分支导致的性能降低？可以从软件和硬件的角度进行分析。
5. 查阅MIPS相关资料，说说其对分支指令是如何进行优化的。
6. 在之前的理想流水线实验中，这些冲突是如何被避免的？

1. 当executeUnit.info.reg\_waddr各位不全为0时，executeUnit.info.reg\_waddr.orR的结果为1。 [↑](#footnote-ref-1)