多周期流水线处理器设计

黄浩睿 2018202143

概述

多周期处理器由以下模块构成：

- 顶层，以及流水线的每一阶段
- 程序计数器（Program Counter）
- 控制器单元（Controller Unit）
- 通用寄存器（General Purpose Registers）
- 转发单元（Forwarding Unit）
- 算术逻辑单元（Arithmetic Logic Unit）
- 乘除法单元（Multiplication Division Unit）
- 指令存储器（Instruction Memory）
- 数据存储器（Data Memory）

从时钟的一个**上升沿**到下一个**上升沿**为一个机器周期。流水线在**上升沿**工作，对寄存器的写入操作在**下降沿**进行。

在时钟上升沿（或下降沿，取决于每个元件的触发时机）时，如果复位信号被设置，则元件的内部状态被复位。

流水线设计

流水线共分为五个阶段。

Fetch 阶段

从指令存储器中获取指令，存入到流水线寄存器中。

Decode 阶段

将指令输入到控制器单元中，并将控制器单元输出的**控制信号**存入到流水线寄存器中。控制信号会沿着流水线寄存器传递到每一阶段。

根据控制信号，读取两个寄存器的值，并通过转发单元尝试从数据旁路获取他们最新的值，并存入到流水线寄存器中。

如果当前指令为分支/跳转指令，则进行分支判断并向 PC 输出跳转信号与跳转到的地址。如果当前指令需要保存返回地址，则将返回地址作为**指令结果**存入到流水线寄存器中。

Execuation 阶段

接受来自上一阶段的寄存器值，并通过转发单元尝试从数据旁路获取他们最新的值，对于运算类指令，寄存器值被输入到算术逻辑单元或乘除法单元中，寄存器值也被存入到流水线寄存器中。

如果当前指令需要算术逻辑单元或乘除法单元的输出结果，则将对应的输出作为**指令结果**存入流水线寄存器中。

Memory 阶段

接受来自上一阶段的寄存器值，并通过转发单元尝试从数据旁路获取他们最新的值，对于读内存类指令，将从内存中读到的值作为**指令结果**写入到流水线寄存器中。对于写内存类指令，控制内存执行写入操作（对于非对齐读写内存类指令，在数据存储器模块中进行特殊处理）。

WriteBack 阶段

接受来自上一阶段的控制信号与**指令结果**，对于产生结果的指令，控制通用寄存器将指令结果写入到寄存器中。

除此之外，每个阶段会判断上个阶段的流水线寄存器中，**指令结果**是否已就绪，如果已就绪则直接写入到本阶段的流水线寄存器中。

数据冒险

控制器单元

与单周期版本相比，控制器加入了用于控制数据旁路的控制信号 —— 即读取到的（每个）寄存器值分别在哪个阶段需要。

如：分支类指令需要读取作为分支判断输入的两个寄存器，并在 Decode 阶段需要这两个寄存器的值；写内存指令需要读取分别作为基址和待写入数据和的两个寄存器，并分别在 Execuation 阶段（计算地址）和 Memory 阶段（执行写入操作）需要这两个寄存器的值。

这样做的意义是，如果指令到达了某个阶段，但该阶段的转发单元的输出结果为，当前指令所读取的某个寄存器的最新值还未就绪（即，需要等待之前的指令执行到更后面的阶段，这些寄存器的最新值才就绪），则可以根据当前阶段是否需要该值来决定是否阻塞流水线 —— 因为，如果当前阶段不需要该值，则可以直接忽略转发单元输出的阻塞信号，继续执行，在下一个阶段继续尝试获取最新的值，直到需要该值的阶段，以给之前的指令更多的执行时间，更可能在不需要阻塞的情况下从旁路获取数据。

转发单元

每一阶段需要寄存器中数据的流水线阶段，都需要放置两个转发单元，分别处理该指令所需要的两个寄存器的数据旁路。

转发单元接受，从该阶段流水线阶段开始，连续三阶段流水线阶段输出的**写入的寄存器编号**、**指令结果**以及**结果是否已就绪**，作为输入值。转发单元会从前往后依次判断，需要转发的寄存器编号，是不是之前某条指令所写入的寄存器编号，如果是，则根据结果是否已就绪，来输出结果或者输出阻塞信号。

存在一种特殊情况，如果某条指令的前两条指令均为转发源，且都在 Memory 阶段结果就绪，且当前指令需要在 Execuation 阶段使用数据，则需要阻塞一个周期，当前一条指令进入 Memory 阶段后，其结果就绪，但更前一条指令在 Memory 阶段产生的结果已经由 WriteBack 阶段处理完成，不再存在于流水线上 —— 也就是说，在阻塞以等待一个寄存器转发源变得就绪的过程中，当前指令失去了另一个寄存器最新值的转发源。

为了避免这种情况，我们在转发单元中存储转发值 —— 当转发源变得就绪时，更新存储的转发值。这个更新操作在时钟**下降沿**进行 —— 由于流水线在**上升沿**工作，所以**下降沿**时流水线寄存器中的结果是稳定的。为了能够在新指令到来时清除状态，我们令 PC 模块输出一个 1 位的信号，每当读取新的指令时该信号翻转（直接判断 PC 改变是不行的，因为有跳转到分支延迟槽中指令，导致 PC 未改变但执行了新的指令的情况）。

阻塞与气泡

当一条指令在某个阶段，由于旁路无法解决的数据冒险，或者结构冒险（如 MDU 在忙），导致无法继续执行时，它会在该阶段**产生阻塞**。如果某个阶段**产生阻塞**，会导致它以及它之前的所有阶段**被阻塞**。被阻塞的阶段不执行任何动作，亦不向其流水线寄存器中写入新的值。在每个阻塞阶段完成后，最远的**产生阻塞**的阶段会产生一个气泡，写入到其流水线寄存器结果中。气泡会使得下一个阶段在下一个周期**被阻塞**。

气泡的作用是，对于已经在某个阶段执行过的指令，当上一个阶段的指令上个周期被阻塞了，导致当前周期该阶段得不到新的指令时，保证该阶段不会重复执行两次该指令（以得到错误的结果）。仅最远的产生阻塞的阶段会产生气泡，而不是每个产生阻塞的阶段都会产生气泡 —— 如果两个阶段同时产生气泡，则非最远被阻塞阶段产生的气泡会导致其之后的一阶段在其指令还未执行的情况下被气泡，导致该指令未能执行完就从流水线上消失。解决此问题只需要判断相邻两阶段（Decode 与 Execuation）同时产生阻塞的情况，因为只有这两个阶段会产生阻塞。

如果一个阶段被气泡阻塞了，则它需要将气泡传递给下一个阶段。

不足之处

通用寄存器的写入与转发单元状态的更新在时钟**下降沿**进行，即两次流水线工作的间隔 —— 这导致了两次流水线工作的间隔不能过短，必须长于前者在**下降沿**时逻辑的延迟。将通用寄存器的写入改变为上升沿进行，并进行旁路转发，并将转发单元改为无状态（通过令 Decode 阶段即使**被阻塞**，也会更新流水线寄存器中的寄存器值来解决上述引入有状态时所解决的问题），即可消除**下降沿**触发的行为，以降低对时钟周期长度的要求。

乘除法单元中，如果正在进行乘除法操作，而新的乘除法操作已经进入了 Execuation 阶段，则可以直接停止之前的乘除法操作，执行新的操作，以消除不必要的等待；如果正在进行乘除法操作，而新的两条指令分别要写入 `HI` 和 `LO` 两个内部寄存器，则之前未完成的乘除法操作也是可以停止的 —— 这需要引入寄存器重命名技术，来判断 `HI` 和 `LO` 是否都被覆写了。