的处理器中，这些操作的典型执行时间从浮点加法的 3 或 4 个周期到整数除法的 64 个周期。为了实现这些指令，我们既需要额外的硬件来执行这些计算，还需要一种机制来协调这些指令的处理与流水线其他部分之间的关系。

实现多周期指令的一种简单方法就是简单地扩展执行阶段逻辑的功能，添加一些整数和浮点算术运算单元。一条指令在执行阶段中逗留它所需要的多个时钟周期，会导致取指和译码阶段暂停。这种方法实现起来很简单，但是得到的性能并不是太好。

通过采用独立于主流水线的特殊硬件功能单元来处理较为复杂的操作，可以得到更好的性能。通常，有一个功能单元来执行整数乘法和除法，还有一个来执行浮点操作。当一条指令进入译码阶段时，它可以被发射到特殊单元。在这个特殊单元执行该操作时，流水线会继续处理其他指令。通常，浮点单元本身也是流水线化的，因此多条指令可以在主流水线和各个单元中并发执行。

不同单元的操作必须同步，以避免出错。比如说，如果在不同单元执行的各个指令之间有数据相关，控制逻辑可能需要暂停系统的某个部分，直到由系统其他某个部分处理的操作的结果完成。经常使用各种形式的转发，将结果从系统的一个部分传递到其他部分，这和前面 PIPE 各个阶段之间的转发一样。虽然与 PIPE 相比，整个设计变得更为复杂，但还是可以使用暂停、转发以及流水线控制等同样的技术来使整体行为与顺序的 ISA 模型相匹配。

2. 与存储系统的接口

在对 PIPE 的描述中，我们假设取指单元和数据内存都可以在一个时钟周期内读或是写内存中任意的位置。我们还忽略了由自我修改代码造成的可能冒险，在自我修改代码中，一条指令对一个存储区域进行写，而后面又从这个区域中读取指令。进一步说，我们是以存储器位置的虚拟地址来引用它们的，这要求在执行实际的读或写操作之前，要将虚拟地址翻译成物理地址。显然，要在一个时钟周期内完成所有这些处理是不现实的。更糟糕的是，要访问的存储器的值可能位于磁盘上，这会需要上百万个时钟周期才能把数据读入到处理器内存中。

正如将在第 6 章和第 9 章中讲述的那样，处理器的存储系统是由多种硬件存储器和管理虚拟内存的操作系统软件共同组成的。存储系统被组织成一个层次结构，较快但是较小的存储器保持着存储器的一个子集，而较慢但是较大的存储器作为它的后备。最靠近处理器的一层是*高速缓存*(cache)存储器，它提供对最常使用的存储器位置的快速访问。一个典型的处理器有两个第一层高速缓存——一个用于读指令，一个用于读和写数据。另一种类型的高速缓存存储器，称为*翻译后备缓冲器*(Translation Look-aside Buffer，TLB)，它提供了从虚拟地址到物理地址的快速翻译。将 TLB 和高速缓存结合起来使用，在大多数时候，确实可能在一个时钟周期内读指令并读或是写数据。因此，我们的处理器对访问存储器的简化看法实际上是很合理的。

虽然高速缓存中保存有最常引用的存储器位置，但是有时候还会出现高速缓存*不命中*(miss)，也就是有些引用的位置不在高速缓存中。在最好的情况中，可以从较高层的高速缓存或处理器的主存中找到不命中的数据，这需要 3～20 个时钟周期。同时，流水线会简单地暂停，将指令保持在取指或访存阶段，直到高速缓存能够执行读或写操作。至于流水线设计，通过添加更多的暂停条件到流水线控制逻辑，就能实现这个功能。高速缓存不命中以及随之而来的与流水线的同步都完全是由硬件来处理的，这样能使所需的时间尽可能地缩短到很少数量的时钟周期。