```
        icode == IRET : valM;
        # Default: Use incremented PC
        1 : valP;
];
```

6. SEQ 小结

现在我们已经浏览了 Y86-64 处理器的一个完整的设计。可以看到，通过将执行每条不同指令所需的步骤组织成一个统一的流程，就可以用很少量的各种硬件单元以及一个时钟来控制计算的顺序，从而实现整个处理器。不过这样一来，控制逻辑就必须要在这些单元之间路由信号，并根据指令类型和分支条件产生适当的控制信号。

SEQ 唯一的问题就是它太慢了。时钟必须非常慢，以使信号能在一个周期内传播所有的阶段。让我们来看看处理一条 ret 指令的例子。在时钟周期起始时，从更新过的 PC 开始，要从指令内存中读出指令，从寄存器文件中读出栈指针，ALU 将栈指针加 8，为了得到程序计数器的下一个值，还要从内存中读出返回地址。所有这一切都必须在这个周期结束之前完成。

这种实现方法不能充分利用硬件单元，因为每个单元只在整个时钟周期的一部分时间内才被使用。我们会看到引入流水线能获得更好的性能。

4.4　流水线的通用原理

在试图设计一个流水线化的 Y86-64 处理器之前，让我们先来看看流水线化的系统的一些通用属性和原理。对于曾经在自助餐厅的服务线上工作过或者开车通过自动汽车清洗线的人，都会非常熟悉这种系统。在流水线化的系统中，待执行的任务被划分成了若干个独立的阶段。在自助餐厅，这些阶段包括提供沙拉、主菜、甜点以及饮料。在汽车清洗中，这些阶段包括喷水和打肥皂、擦洗、上蜡和烘干。通常都会允许多个顾客同时经过系统，而不是要等到一个用户完成了所有从头至尾的过程才让下一个开始。在一个典型的自助餐厅流水线上，顾客按照相同的顺序经过各个阶段，即使他们并不需要某些菜。在汽车清洗的情况中，当前面一辆汽车从喷水阶段进入擦洗阶段时，下一辆就可以进入喷水阶段了。通常，汽车必须以相同的速度通过这个系统，避免撞车。

流水线化的一个重要特性就是提高了系统的*吞吐量*(throughput)，也就是单位时间内服务的顾客总数，不过它也会轻微地增加*延迟*(latency)，也就是服务一个用户所需要的时间。例如，自助餐厅里的一个只需要甜点的顾客，能很快通过一个非流水线化的系统，只在甜点阶段停留。但是在流水线化的系统中，这个顾客如果试图直接去甜点阶段就有可能招致其他顾客的愤怒了。

4.4.1　计算流水线

让我们把注意力放到计算流水线上来，这里的“顾客”就是指令，每个阶段完成指令执行的一部分。图 4-32a 给出了一个很简单的非流水线化的硬件系统例子。它是由一些执行计算的逻辑以及一个保存计算结果的寄存器组成的。时钟信号控制在每个特定的时间间隔加载寄存器。CD 播放器中的译码器就是这样的一个系统。输入信号是从 CD 表面读出的位，逻辑电路对这些位进行译码，产生音频信号。图中的计算块是用组合逻辑来实现的，意味着信号会穿过一系列逻辑门，在一定时间的延迟之后，输出就成为了输入的某个函数。