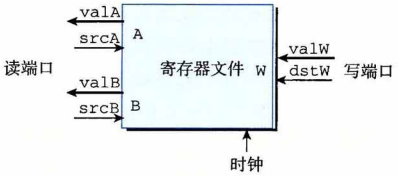
存器是作为电路不同部分中的组合逻辑之间的屏障。每当每个时钟到达上升沿时，值才会从寄存器的输入传送到输出。我们的 Y86-64 处理器会用时钟寄存器保存程序计数器(PC)、条件代码(CC)和程序状态(Stat)。

图 4-16　寄存器操作。寄存器输出会一直保持在当前寄存器状态上，直到时钟信号上升。当时钟上升时，寄存器输入上的值会成为新的寄存器状态

下面的图展示了一个典型的寄存器文件：



寄存器文件有两个读端口(A 和 B)，还有一个写端口(W)。这样一个多端口随机访问存储器允许同时进行多个读和写操作。图中所示的寄存器文件中，电路可以读两个程序寄存器的值，同时更新第三个寄存器的状态。每个端口都有一个地址输入，表明该选择哪个程序寄存器，另外还有一个数据输出或对应该程序寄存器的输入值。地址是用图 4-4 中编码表示的寄存器标识符。两个读端口有地址输入 srcA 和 srcB(“source A”和“source B”的缩写)和数据输出 valA 和 valB(“value A”和“value B”的缩写)。写端口有地址输入 dstW(“destination W”的缩写)，以及数据输入 valW(“value W”的缩写)。

虽然寄存器文件不是组合电路，因为它有内部存储。不过，在我们的实现中，从寄存器文件读数据就好像它是一个以地址为输入、数据为输出的一个组合逻辑块。当 srcA 或 srcB 被设成某个寄存器 ID 时，在一段延迟之后，存储在相应程序寄存器的值就会出现在 valA 或 valB 上。例如，将 srcA 设为 3，就会读出程序寄存器%rbx 的值，然后这个值就会出现在输出 valA 上。

向寄存器文件写入字是由时钟信号控制的，控制方式类似于将值加载到时钟寄存器。每次时钟上升时，输入 valW 上的值会被写入输入 dstW 上的寄存器 ID 指示的程序寄存器。当 dstW 设为特殊的 ID 值 0xF 时，不会写任何程序寄存器。由于寄存器文件既可以读也可以写，一个很自然的问题就是“如果我们试图同时读和写同一个寄存器会发生什么?”答案简单明了：如果更新一个寄存器，同时在读端口上用同一个寄存器 ID，我们会看到一个从旧值到新值的变化。当我们把这个寄存器文件加入到处理器设计中，我们保证会考虑到这个属性的。

处理器有一个随机访问存储器来存储程序数据，如下图所示：