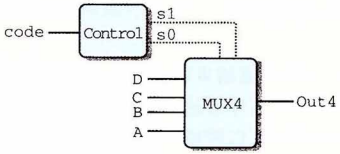
如下图所示：



在这个电路中，两位的信号 code 就可以用来控制对 4 个数据字 A、B、C 和 D 做选择。根据可能的 code 值，可以用相等测试来表示信号 s1 和 s0 的产生：

```
bool s1 = code == 2 || code == 3;
bool s0 = code == 1 || code == 3;
```

还有一种更简洁的方式来表示这样的属性：当 code 在集合{2，3}中时 s1 为 1，而 code 在集合{1，3}中时 s0 为 1：

```
bool s1 = code in { 2, 3 };
bool s0 = code in { 1, 3 };
```

判断集合关系的通用格式是：

$$iexpr \text{ in } \{iexpr_1, iexpr_2, \cdots, iexpr_k\}$$

这里被测试的值 $iexpr$ 和待匹配的值 $iexpr_1 \sim iexpr_k$ 都是整数表达式。

4.2.5　存储器和时钟

组合电路从本质上讲，不存储任何信息。相反，它们只是简单地响应输入信号，产生等于输入的某个函数的输出。为了产生时序电路(sequential circuit)，也就是有状态并且在这个状态上进行计算的系统，我们必须引入按位存储信息的设备。存储设备都是由同一个时钟控制的，时钟是一个周期性信号，决定什么时候要把新值加载到设备中。考虑两类存储器设备：

- 时钟寄存器(简称寄存器)存储单个位或字。时钟信号控制寄存器加载输入值。
- 随机访问存储器(简称内存)存储多个字，用地址来选择该读或该写哪个字。随机访问存储器的例子包括：1)处理器的虚拟内存系统，硬件和操作系统软件结合起来使处理器可以在一个很大的地址空间内访问任意的字；2)寄存器文件，在此，寄存器标识符作为地址。在 IA32 或 Y86-64 处理器中，寄存器文件有 15 个程序寄存器(%rax～%r14)。

正如我们看到的那样，在说到硬件和机器级编程时，“寄存器”这个词是两个有细微差别的事情。在硬件中，寄存器直接将它的输入和输出线连接到电路的其他部分。在机器级编程中，寄存器代表的是 CPU 中为数不多的可寻址的字，这里的地址是寄存器 ID。这些字通常都存在寄存器文件中，虽然我们会看到硬件有时可以直接将一个字从一个指令传送到另一个指令，以避免先写寄存器文件再读出来的延迟。需要避免歧义时，我们会分别称呼这两类寄存器为“硬件寄存器”和“程序寄存器”。

图 4-16 更详细地说明了一个硬件寄存器以及它是如何工作的。大多数时候，寄存器都保持在稳定状态(用 x 表示)，产生的输出等于它的当前状态。信号沿着寄存器前面的组合逻辑传播，这时，产生了一个新的寄存器输入(用 y 表示)，但只要时钟是低电位的，寄存器的输出就仍然保持不变。当时钟变成高电位的时候，输入信号就加载到寄存器中，成为下一个状态 y，直到下一个时钟上升沿，这个状态就一直是寄存器的新输出。关键是寄