十六进制数 9A48Ch 可以表示为如下形式：

$$\begin{aligned}9A48Ch = {} & 9 \times 10000h + \\ & A \times 1000h + \\ & 4 \times 100h + \\ & 8 \times 10h + \\ & C \times 1h\end{aligned}$$

这个数用 16 的乘方表示可以写为：

$$\begin{aligned}9A48Ch = {} & 9 \times 16^4 + \\ & A \times 16^3 + \\ & 4 \times 16^2 + \\ & 8 \times 16^1 + \\ & C \times 16^0\end{aligned}$$

也可以将 16 的幂进一步展开，写为如下形式：

$$\begin{aligned}9A48Ch = {} & 9 \times 65{,}536 + \\ & A \times 4096 + \\ & 4 \times 256 + \\ & 8 \times 16 + \\ & C \times 1\end{aligned}$$

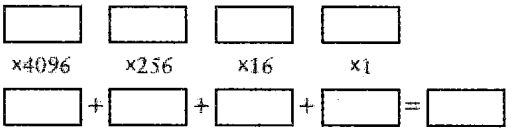
我们可以仔细想想，如果只把一个数字（比如 9，A，4，8 和 C 中的任何一个）单独列举出来，而且不使用任何下标来指明其进制数，这种做法并不会产生二义性。无论在十进制还是十六进制下，一个单独的 9 仅代表 9；而 A 的出现也说明了它本身是一个十六进制数——等价于十进制中的 10。

我们可以拿起笔和纸进行演算了，把十六进制数转换成十进制数：

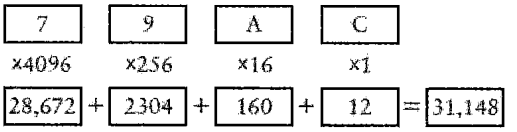
$$\begin{aligned}9A48Ch = {} & 9 \times 65{,}536 + \\ & 10 \times 4096 + \\ & 4 \times 256 + \\ & 8 \times 16 + \\ & 12 \times 1\end{aligned}$$

运算的最后结果是 631,948。这就是一个十六进制数转换成为十进制数的完整过程。

下面给出了一种模板，它可以帮助我们把 4 位十六进制转换成为十进制。

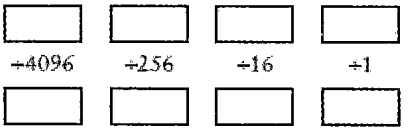


下面我们来看一个例子，把 79ACh 转化成为十进制数。需要牢记在心的就是 A 和 C 分别代表 10 和 12。

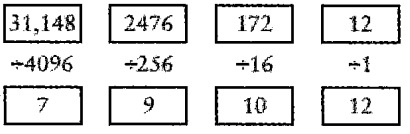


十进制数转换为十六进制数通常涉及除法运算。我们知道，小于或等于 255 的数用 1 个字节就足以表示，也就是两个十六进制数。如何来求出这两个数呢？通常可以用这个数除以 16，分别得到商和余数，商作为结果保留，而余数则作为下次运算的除数。我们举一个先前讲过的例子来进一步阐述运算法则：十进制数 182，除以 16，商为 11（在十六进制中表示为 B），余数为 6，所以它的十六进制为 B6h。

如果被转化的十进制数小于 65,536，那么就可以使用少于 4 位的十六进制数来表示。下面给出一个十进制向十六进制数转化的一个模板。



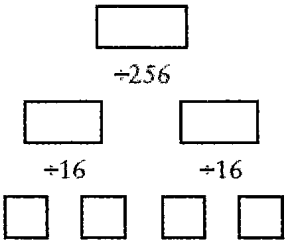
使用时首先把十进制写到左上角的方框里。这个方框代表着第一个被除数，然后除以第一个除数 4096，得到的商放到被除数所对应的下面的方框里，而将余数放到被除数右边的方框里。这时将余数作为新的被除数去除以 256。利用这个规则，通过反复迭代就可以得到最终结果。下面这幅图向我们展示了十进制数 31,148 转换成十六进制数的过程。



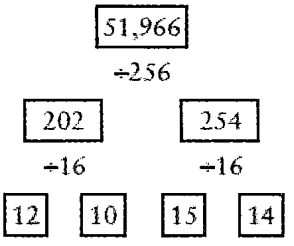
显而易见，十进制数的 10 和 12 代表着十六进制中的 A 和 C，计算得到的最后结果就是 79ACh。

这个方法存在一个小小的问题，那就是如果使用计算器帮助你进行除法运算，它不会显示每步运算得到的余数是多少。如果让计算器去运算 31,148 除以 4096，它的屏幕上只会出现 7.6044921875。我们可以用倒推的方法计算余数，用 4096 乘以 7（得到 28,672），从 31,148 中减去它，这是一种办法。也可以用 4096 乘以 0.6044921875，也就是计算器运算结果的小数部分（现在市面上很多计算器已经具备了十进制和十六进制数之间的转换功能）。

还有一种转换小于 65,535 的十六进制数的方法，首先我们把原数通过除以 256 的方式将其分为两个字节。接下来对于每个字节，再分别除以 16。下图是运算过程使用到的模板。



我们采用自顶向下的方式。每一次除法完成后，就将其得到的商放入除数左下方的方框里，而余数进入右边的方框里。下图举例说明了十进制数 51,966 的转换过程。



最后我们得到了四个 1 位的十六进制数字，其十进制值分别是 12、10、15 和 14，转换过来就是 CAFE，无论怎么看它都更像一个单词，很难想象它其实是一个数字（我想应该没人愿意点一杯叫做 56,495 的东西，然后把它当作咖啡喝下去吧）!

对于每种计数系统，我们都可以描绘出相应的操作运算表，下面是十六进制的加法运算表。

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

使用这张表可以方便地仿照一般的加法运算来对十六进制数进行加运算，如下例所示：

```
  4A3378E2
+ 877AB982
----------
  D1AE3264
```

回忆一下第 13 章的内容，我们讨论过可以用一个 2 的补数来表示与其相对应的负数。如果我们处理的是带符号的 8 位二进制数，那么所有负数的最高位都为 1。在十六进制系统中，最高位为 8、9、A、B、C、D、E 或 F 的两位带符号数都是负数，因为这些十六进制数对应的二进制数的最高位为 1。例如 99h 可以表示无符号的十进制数 153（你必须清楚它是单字节的无符号数），也可以表示十进制的–103（这时它被看做有符号数）。

奇妙的是，99h 这个十六进制字节从某种意义上来讲，也代表着十进制的 99！我们都非常想知道原因，不过这种说法似乎和先前所学到的所有东西相抵触。我会在第 23 章中进行详细解释。但在此之前，我们必须先讨论一下存储器方面的知识。

16 存储器组织

每天清晨，我们将自己从沉睡中唤醒，这时大脑的空白会很快被记忆填充。我们立刻会意识到自己身在何方，最近做了些什么事情，有什么计划和打算。有的事情我们很快就能想起来，但有时，我们大脑处于失忆状态，有那么几分钟发现自己什么都想不起来（就拿我来说，有时我就是想不起来怎么我上床时还穿着袜子），但总的来说，我们总是能够与自己的过去保持足够的连续性，继续新的生活，展开人生新的一页。

显然，人类的记忆似乎没有什么规律。仔细回想高中的几何课，或许你一下子就能想到是谁坐在你前面，或许你清晰地记得当老师讲到 QED（quod erat demonstrandum，证明与推论）这个概念的时候消防演习开始了。

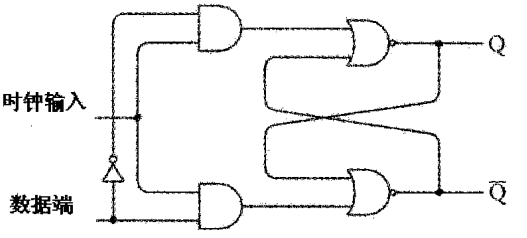
人类的记忆也并非能面面俱到。书面记录这种技术的引入，从某种层面来讲，就是为了弥补人类记忆容易遗漏这一缺陷。或许在某天夜里凌晨三点，你从床上一跃而起，脑海中突然出现某个剧本的绝妙灵感。你立刻抓起床边提前预备的笔和纸，将它们全部记下防止遗忘，然后才安然入睡。一觉醒来的清晨，再次浏览这个绝妙的灵感，一个新的剧本构想跃然纸上（剧本的内容就是“一次邂逅，汽车追尾与爆炸”?仅此而已吗？）或许远不如此。

我们总是将需要的记住的内容事先记下来，在需要时拿出来阅读；习惯于将可能用

到的事物先存起来，在需要时将它们取出。从技术角度来讲，这个过程称为先存储后访问。存储器的职责和作用就在于此，它负责保障这两个过程之间信息完好无损。我们每次存储信息都要利用不同种类的存储器。比如，保存文本信息的不二之选就是纸张，而磁带则更适于存储音乐和电影。

电报继电器（Telegraph Relays）——以一定形式组织起来构成逻辑门，然后再形成触发器——同样具备保存信息的能力。在前面章节中我们讨论过，一个触发器可以对 1 位信息进行存储。这样的存储能力要存储一大堆的信息还远远不够，但它却为我们达到目标迈出了坚实的一步。其实知道了如何存储 1 位信息，很容易就可以想象出如何存储 2 位、3 位或更多位信息。

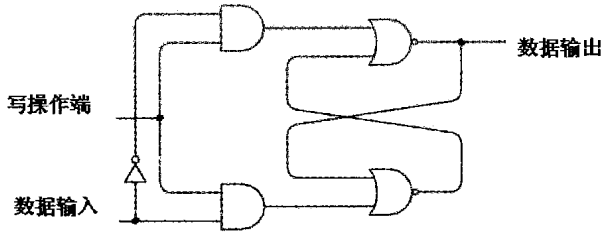
在第 14 章的学习过程中，我们一起讨论过由一个反向器、两个与门和两个或非门构成的 D 型电平触发器，如下图所示。



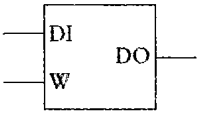
当时钟输入为 1，Q 端输出与数据端输入保持一致。但当时钟输入跳变为 0 时，Q 端输出将保持数据端最后一次的输入。除非时钟输入再次还原为 1，之后的数据端输入不会影响输出。此触发器的真值表如下。

输入		输出	
D	Clk	Q	$\overline{Q}$
0	1	0	1
1	1	1	0
X	0	Q	$\overline{Q}$

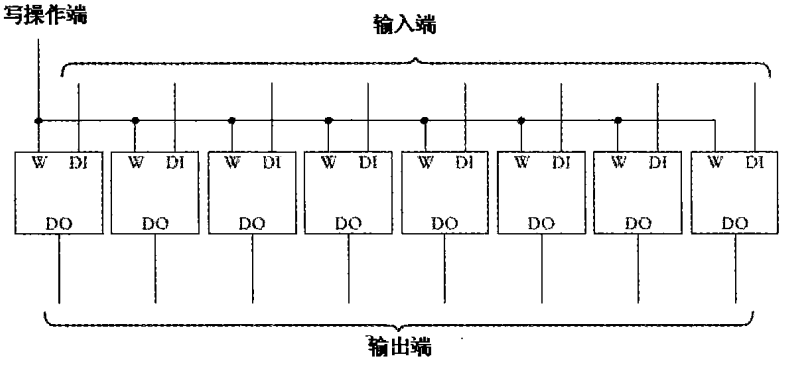
在第 14 章的讨论中，这种触发器可以由两种不同特性的电路来实现，而在本章我们仅选择其中一种——目的就只是为了保存 1 位信息。为了更加清楚地表述，我们给输入和输出端重新命名，使其名称与功能相符，如下图所示。



从结构上来讲，这套电路与先前所学到的是同一种触发器，只是命名的方式不尽相同，现在 Q 输出端被称为数据输出端（Data Out），时钟输入端（在第 14 章叫做保持位）命名为写操作端（Write）。就像信息可以被记录在纸上一样，写操作端的信号同样使得数据输入（Data In）信号被写入（Written Into），也可以称之为被存储（stored）到电路中。一般情况下，如果写操作端为 0，则数据输入信号的状态对输出无影响。而当我们想把数据输入信号存储在触发器中时，可以把写入信号应先置 1 后置为 0。在第 14 章讲到过，这种类型的电路也被称为锁存器，因为存储进去的数据就好像被锁住了一样。下面给出了 1 位锁存器简化框图，框图未画出其内部结构中的部件。

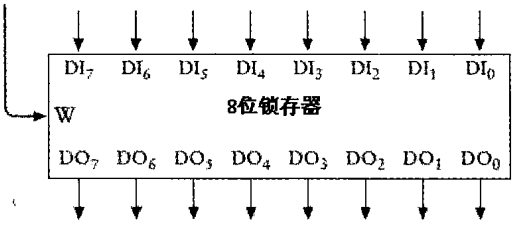


我们很容易想到如何把多个 1 位锁存器组织成为多位锁存器，所要做的就是把写操作端的信号连接到系统中，就像下面这样。

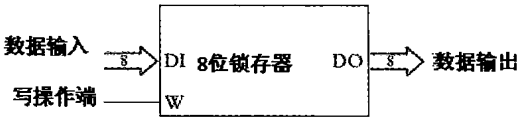


图中显示的 8 位锁存器其输入和输出端各有 8 个。另外还包括一个写操作端，在非工作状态下一般为 0。如果要把一个 8 位二进制数存储在锁存器中，首先要把写操作端置 1，

然后置 0。我们同样可以把这个锁存器以框图的形式表现出来，就像下面这样。



为了和先前提到的 1 位锁存器保持一致，我们将它可以画成下面这种形式。



还有另一种方法集成 8 个 1 位锁存器，但其结构并不像上面的这样直观。假设我们只想用一个数据输入和输出信号端，而且希望锁存器能将输入信号数据分 8 次独立存储，这个任务可以在长达一天内完成，或者可能迅速在下一分钟内搞定。最后一项要求就是我们还希望能够通过观察数据输出信号端确定实际的 8 位输出。

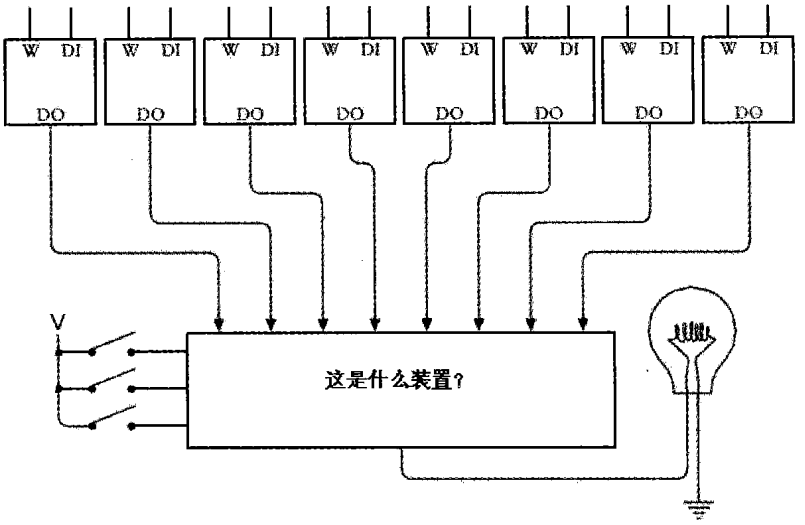
换句话说，在这种锁存器中我们只想存储 8 个单独的比特，而不是存储 1 个 8 位二进制数。

为什么会有这种需求呢？原因可能在于仅有一个灯泡！

我们知道现在所需要的是 8 个 1 位锁存器。先不去考虑数据如何存储在这些锁存器中，把重点放在如何用一个灯泡来确定锁存器的数据输出信号。最简单的方法就是把这个灯泡依次连接到每个锁存器上，分若干次来测试各个锁存器的输出，但是我们要追求更加自动化的方法。用开关来选择想要检查的锁存器是一个好的办法。

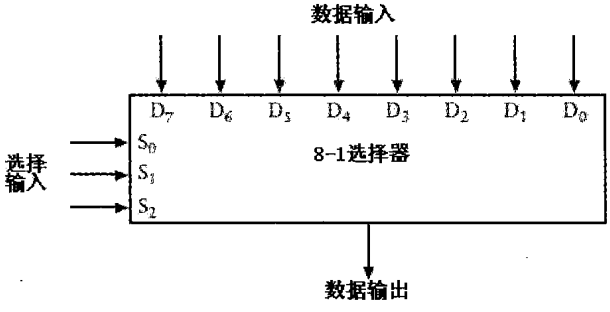
究竟需要多少个开关才能解决问题呢？我们可以把这个过程进一步抽象，问题变成了怎么样从 8 个物体中选出一个我们想要的，我们需要 3 个开关。这是因为通过 3 个开关连通与闭合的排列组合，总共可表示出 8 个不同的值：000、001、010、011、100、101、110 和 111。

现在我们手头上已有 8 个 1 位锁存器、3 个开关、1 个灯泡，此外在开关和灯泡之间还有另外一种装置，如下图所示。



这个“额外装置”就是图中的神秘盒子，顶部带有 8 个输入端，左侧也带有 3 个输入端。通过三个开关的闭合和断开，对顶部的输入进行 8 选 1 操作，输出结果被传递到其底部连接的灯泡，使其发光。

上图标注的“这是什么装置？”到底是什么呢？我们先前曾碰见过类似的东西，当时讨论的装置没有这么多的输入端。这种装置曾出现在第 14 章中第一个改进的加法机的电路中。在该电路中要在一行开关和一个锁存器的输出之间选择一个，作为加法器的输入，当时称其为 2-1 选择器。而我们现在所需要的正是 8-1 数据选择器（8-Line-to-1-Line Data Selector）。

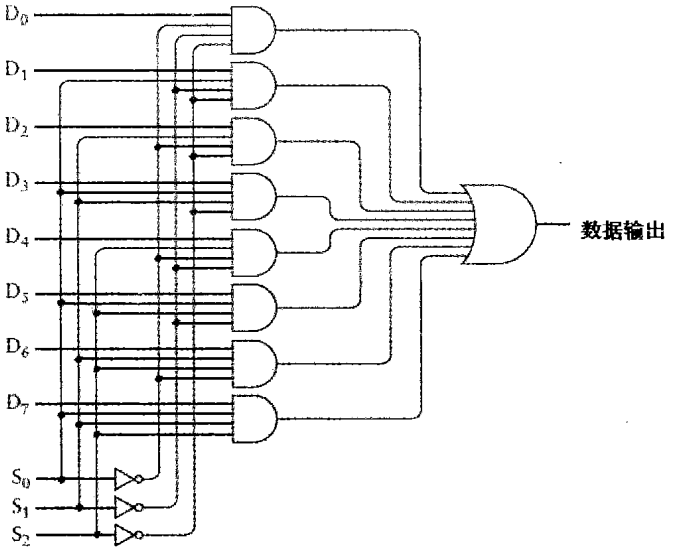


8-1 选择器有 8 个数据输入端（在其顶部），以及 3 个选择输入端（在其左侧）。选择输入端的功能就是选择一个输入端数据，然后使其在输出端输出。如果选择输入端为 000，

则将 D_0 锁存器的值输出；若选择端为 111，则 D_7 锁存器的值将被输出；若选择端为 101，则相应地输出 D_5 的值。系统的真值表如下所示。

输入			输出
S_2	S_1	S_0	Q
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

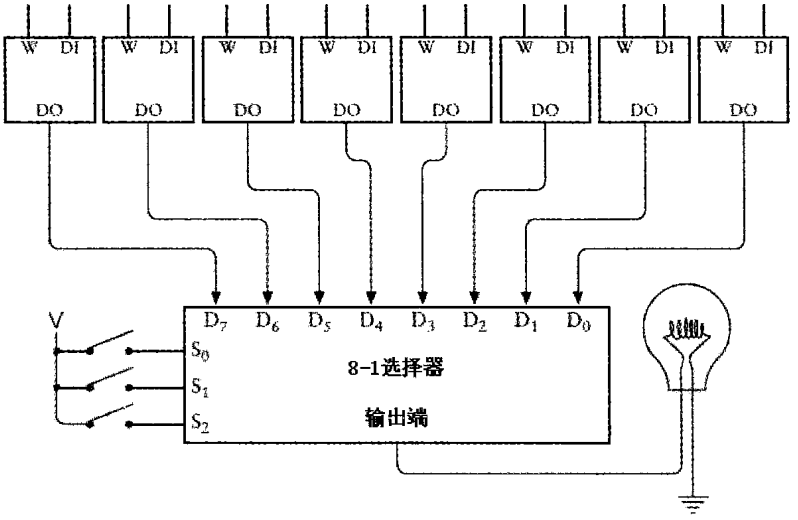
8-1 选择器主要组成部件为：三个反向器、八个 4 端口输入与门、一个 8 端口输入或门，系统的组织结构如下图所示。



这个电路看上去线路密布，要理解它是如何工作的，最好方式就是一起来看一个例子。假设 S_2 初始化为 1，S_1 初始化为 0，S_0 初始化为 1。从顶部开始的第 6 个与门的输入由 S_0、$\overline{S_1}$、S_2 组成，初始状态下它们全为 1。其余与门的这三项输入数据都与第 6 个与

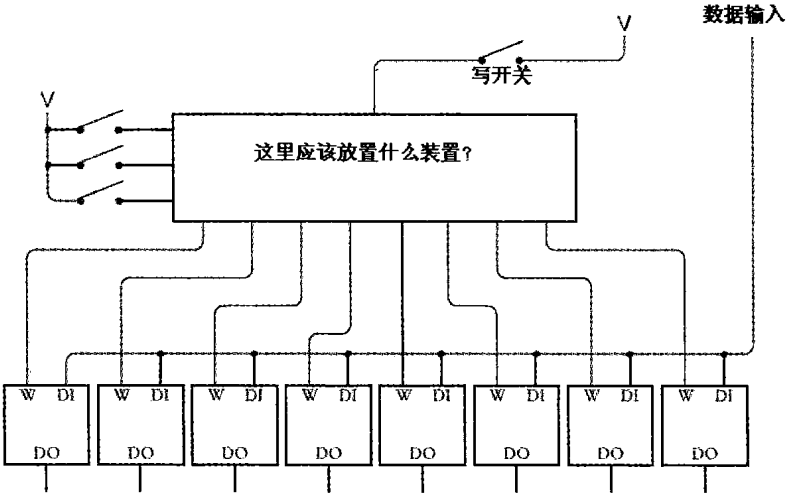
门不尽相同，这使得其余与门输出全部为 0。若 D_5 变为 0 意味着第 6 个与门输出为 0；反之第 6 个与门输出则为 1。对最右边的或门也可以按照同样的方式理解。我们可以总结出下面这个结论：若选择端为 101，则数据输出端与 D_5 的输出保持一致。

让我们重新理一下思路，想想自己究竟要干什么。我们的目的是通过某种方式连接 8 个 1 位锁存器，使自己能够从一个输入信号端写入数据，还能从一个输出信号端鉴别出数据。现在我们已经成功地使用了一个 8-1 选择器对 8 个锁存器进行了选择操作，并将相应锁存器的数据输出，下面是电路的结构图。



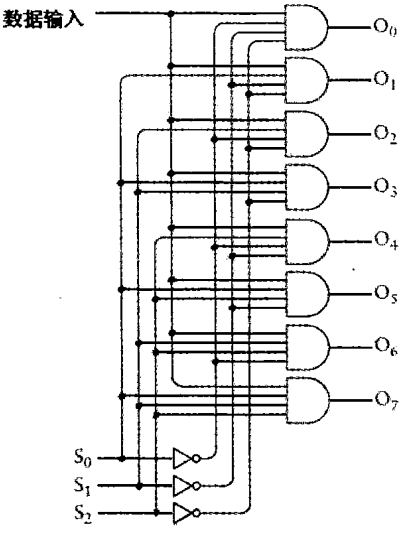
到这里我们只走完了长征的一半。既然输出端已经满足了要求，现在把注意力集中到输入端。

输入端包括了数据输入信号及写操作信号。可以把所有数据输入信号在锁存器的输入端连接在一起。但 8 个写入信号是不可以连在一起的，因为我们很可能要向每个锁存器依次写入数据。除此之外还需要一个独立的写入信号，它能被路由到任意（且唯一）的锁存器上，系统的结构可用下图表示。



为了能圆满完成任务，我们需要另外一款电路元件，而且这款元件与 8-1 选择器功能类似，但它的作用正好相反。我们所说的正是 3-8 译码器（3-to-8 Decoder）。前面的章节中我们曾学习过一个简易的数据译码器（Data Decoder）——在第 11 章中为了选择喜欢的猫咪的毛色，我们把开关以一定方式进行连接使其具有选择功能。

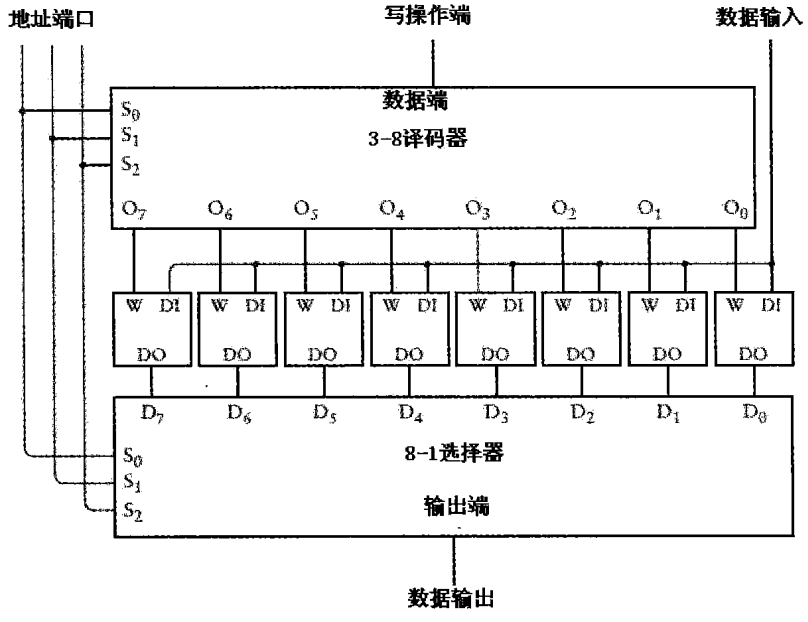
3-8 译码器的输出端口共有 8 个。在任何时刻，译码器只会有一个锁存器的输出为 1，其余均为 0。每一个输出端的结果都是由 S_0、S_1、S_2 这三个信号的排列组合决定的。而数据的输出和输入一致，如下图所示。



我想再次强调一遍：注意从上往下数的第 6 个与门，它的输入包括 S_0、$\overline{S_1}$、S_2。没有任何一个与门具有和它相同的三个输入。在这种情况下，如果选择输入端为 101，则除了 O_5 要根据情况进行判定外，其余与门输出都为 0。这个时候，若数据端输入为 0，则 O_5 随之输出为 0；相应的，若数据端输入为 1，则 O_5 输出为 1。译码器的逻辑表可以如下表所示。

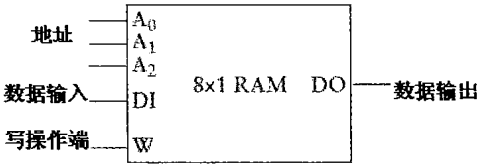
输入			输出							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_4	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	Data
0	0	1	0	0	0	0	0	0	Data	0
0	1	0	0	0	0	0	0	Data	0	0
0	1	1	0	0	0	0	Data	0	0	0
1	0	0	0	0	0	Data	0	0	0	0
1	0	1	0	0	Data	0	0	0	0	0
1	1	0	0	Data	0	0	0	0	0	0
1	1	1	Data	0	0	0	0	0	0	0

将 8 个锁存器加入到电路就形成了完整的系统。



值得注意的是，译码器和选择器具有相同的选择信号，在上图中这三个信号一起被称为地址端口（Address）。地址的作用就像我们平时使用的邮箱号，长度为三位的地址决定了 8 个锁存器中的哪一个将被引用。在 3-8 译码器的输入端，地址起到了决定哪些锁存器可以被写操作端的信号触发来保存数据的作用。在输出端（图的下半部分），8-1 选择器通过地址来选择 8 个锁存器中的一个，最后将其输出。

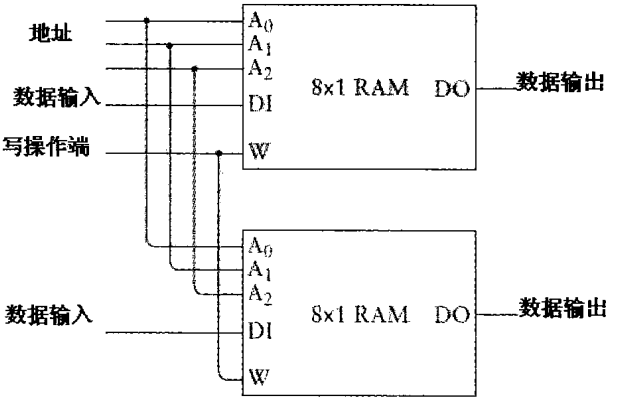
这种配置下的锁存器在有的资料中也被称为读/写存储器（read/write memory），但更普遍的叫法是随机访问存储器（Random Access Memory），或 RAM（和单词 animal 发音类似）。可以认为我们讨论的这种存储器是可存储 8 个独立比特的 RAM，它的简化结构图如下所示。



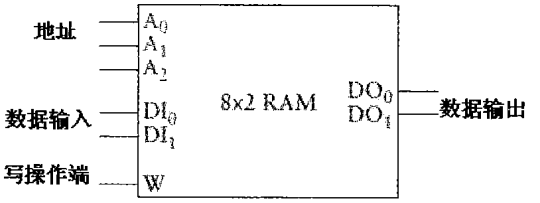
上图所示的电路之所以能够被称为存储器是因为它可以保存信息。而能够被称为读/写存储器是因为它不仅可以在每个锁存器中存储新的数据（可以把这种功能称为写数据），而且我们还可以检查每个锁存器都保存了什么数据（可以把这种功能称为读数据）。之所以可以被称为随机访问存储器，是因为读写操作很自由，我们只需要改变地址及相关的输入，就可以从 8 个锁存器中读出或写入需要的数据。相比于其他的顺序型的存储器——这种存储器在使用时有一定的限制，如果想要读取地址为 101 的数据，必须先把地址为 100 的数据读取出来。

将 RAM 进行特殊的配置可形成 RAM 阵列（Array），我们所讨论的这种 RAM 阵列以 8×1（读做 8 乘 1）的方式组织起来。阵列以 1 比特作为存储单位，共存储 8 个单位的数据。所以这个 RAM 阵列中能存储的位数等于 8 与 1 的乘积。

RAM 阵列的组合形式多种多样。比如我们可以通过共享地址的方式可以把两个 8×1 的 RAM 阵列连接起来，如下图所示。



我们把这两个 8×1 的 RAM 阵列的地址和输出都分别看成一个整体，这样就得到了一个 8×2 的 RAM 阵列，如下图所示。



这个 RAM 阵列可存储的二进制数依然是 8 个，但每个数的位宽为 2 位。

我们还可以把两个 8×1 的 RAM 阵列看做是两个锁存器，使用一个 2-1 选择器和一个 1-2 译码器就可以把它们按照单个锁存器连接方式进行集成，下面给出了这种方案的电路图。