接下来要把 Z80 PIO 的 B/$\overline{A}$ 和 C/$\overline{D}$ 引脚分别连接到 Z80 CPU 的地址总线引脚 A0 和 A1 上。若表示 IC 引脚作用的代号上划有横线，则表示通过赋予该引脚 0（0V）可使之有效，反之若没有横线，则表示通过赋予该引脚 1（+5V）可使之有效。因此若赋予 B/$\overline{A}$ 引脚 1 则表示选中 B，反之赋予 0 则表示选中 A。同样地，若赋予 C/$\overline{D}$ 引脚 1 则表示选中的是 C（C 即 Control，表示控制模式）；反之赋予 0 则表示选中的是 D（D 即 Data，表示数据模式）。

通过 Z80 CPU 的 A0～A7（00000000～11111111 共 256 个地址）地址总线引脚可以选择内存（TC5517）中的存储单元。同样地，使用 Z80 CPU 的 A0～A1（00～11 共 4 个地址）地址总线引脚也可以选择 I/O（Z80 PIO）中的寄存器。

Z80 CPU 的 A8～A15 地址总线引脚尚未使用，所以什么都不连接。在电路图中可以用代号 NC（No Connection，未连接）表示引脚什么都不连接。IC 上的引脚有些只用于输出，有些只用于输入，还有些是输入输出两用的。对于只用于输出的引脚，不需要使用时的处理方法是这个引脚什么都不连接；而对于只用于输入或输入输出两用的引脚，不需要使用时的处理方法则是把这个引脚上的电压固定成是 +5V 或 0V。

2.5 连接时钟信号

正如前文所述，Z80 CPU 和 Z80 PIO 的运转离不开时钟信号。为了传输时钟信号，就需要把时钟发生器的 8 号引脚和 Z80 CPU 的 CLK（CLK 即 Clock，时钟）引脚、Z80 PIO 的 CLK 引脚分别连接起来。时钟发生器的 8 号引脚与 +5V 之间的电阻用于清理时钟信号。

再插入一段题外话。诸位可以把 Z80 CPU 和 Z80 PIO 在时钟信号

下运转的情景，想象成是它们在跟随着滴答滴答响的时钟同步做动作。据说 19 世纪英国的查尔斯・巴贝奇（Charles Babbage）曾向制造计算机的原型——分析机发起过挑战。分析机由齿轮组成，因当时科技水平的限制并未制造完成。可是如果把分析机改用电子元件制造出来的话，就是今天的计算机。

2.6　连接用于区分读写对象是内存还是 I/O 的引脚

至此，我们已经先后把 Z80 CPU 连接到了 TC5517 和 Z80 PIO 上，这两次连接都使用了地址总线引脚 A0 和 A1。如果仅仅这样连接，就会导致一个问题，当地址的最后两位是 00、01、10 和 11 时，CPU 就无法区分访问的是 TC5517 中的存储单元，还是 Z80 PIO 中的寄存器了。

Z80 CPU 上的 $\overline{\text{MREQ}}$（即 Memory Request，内存请求）引脚和 $\overline{\text{IORQ}}$（即 I/O Request，I/O 请求）引脚解决了这个问题。当 Z80 CPU 和内存之间有数据输入输出时，$\overline{\text{MREQ}}$ 引脚上的值是 0，反之则是 1。当 Z80 CPU 和 I/O 之间有数据输入输出时，$\overline{\text{IORQ}}$ 引脚上的值是 0，反之则是 1。

若把 TC5517 的 $\overline{\text{CE}}$（即 Chip Enable，选通芯片）引脚设成 0，则 TC5517 在电路中被激活，若设成 1 则从电路中隔离，因为此时 TC5517 进入了高阻抗状态，所以即便它上面的引脚已经接入了电路也不会接收任何电信号。在 Z80 PIO 中，则是通过将 $\overline{\text{CE}}$ 引脚和 $\overline{\text{IORQ}}$ 引脚同时设为 0 或 1，来达到与 TC5517 的 $\overline{\text{CE}}$ 引脚相同的效果。若同时设为 0，则 Z80 PIO 在电路中被激活，若同时设为 1 则从电路中隔离（之所以使用两个引脚是因为这样更适合使用了多个 I/O 的情况）。

按照上面的讲解，下面需要把 Z80 CPU 的 $\overline{\text{MREQ}}$ 引脚连接到 TC5517 的 $\overline{\text{CE}}$ 引脚上。然后把 Z80 CPU 的 $\overline{\text{IORQ}}$ 引脚连接到 Z80 PIO 的 $\overline{\text{CE}}$ 引脚和 $\overline{\text{IORQ}}$ 引脚上。请诸位先用红铅笔把这些引脚分别连接起来吧。

对内存和 I/O 而言，还必须要分清 CPU 是要输入数据还是输出数据。为此就要用到 Z80 CPU 的 $\overline{RD}$ 引脚（即 Read，表示输入，为 0 时执行输入操作）和 $\overline{WR}$ 引脚（即 Write，表示输出，为 0 时执行输出操作）了。请把这两个引脚与 TC5517 上同名的引脚连接起来。Z80 PIO 虽然只有 $\overline{RD}$ 引脚，但由于数字 IC 引脚上的值要么是 0 要么是 1，所以只用 1 个 $\overline{RD}$ 引脚也能区分是输入还是输出，0 的话是输入，1 的话就是输出（如表 2.2 所示）。

表 2.2　与读写内存、I/O 相关的引脚上的值

Z80 CPU 的操作	$\overline{MREQ}$ 引脚	$\overline{IORQ}$ 引脚	$\overline{RD}$ 引脚	$\overline{WR}$ 引脚
从内存输入	0	1	0	1
向内存输出	0	1	1	0
从 I/O 输入	1	0	0	1
向 I/O 输出	1	0	1	0

2.7　连接剩余的控制引脚

CPU、内存、I/O 中不但有地址总线引脚、数据总线引脚，还有其他引脚，通常把这些引脚统称为“控制引脚”。之所以这样命名是因为这些引脚上输入输出的电信号具有控制 IC 的功能。现在 Z80 CPU 上只剩下 9 个控制引脚没有连接了，那么就再加把劲，继续用红铅笔把它们也连接到电路中吧。

首先把 Z80 CPU 的 $\overline{M1}$ 引脚（即 Machine Cycle 1，机器周期 1）和 $\overline{INT}$ 引脚（即 Interrupt，中断）与 Z80 PIO 上标有相同代号的引脚连接起来。$\overline{M1}$ 是用于同步的引脚，$\overline{INT}$ 引脚是用于从 Z80 PIO 向 Z80 CPU 发出中断请求的引脚。所谓中断就是让 CPU 根据外部输入的数据执行特定的程序。有关中断的详细内容将在第 4 章介绍，这里只需要先记住 I/O 可以中断 CPU 对程序的处理流程就可以了。

一旦把 Z80 CPU 的 $\overline{\text{RESET}}$ 引脚（即 Reset，重置）上的值先设成 0 再还原成 1，CPU 就会被重置，重新从内存 0 号地址上的指令开始顺序往下执行。重置 CPU 可以通过按键开关完成。按键开关需要经过电阻接在 +5V 和 0V 之间。请仔细地观察这一部分的电路图，可以看出 $\overline{\text{RESET}}$ 引脚上平时是 +5V（即 1）。当按下按键开关时，$\overline{\text{RESET}}$ 引脚就变成了 0V（即 0），而放开按键开关后又会回到 +5V（即 1）。电阻是为了防止短路而加入的，否则一旦按下了按键开关，+5V 和 0V 就会直接接到一起发生短路。像这样通过加入电阻把 +5V 和 0V 连接起来的方法在电路图中随处可见（如图 2.8 所示）。

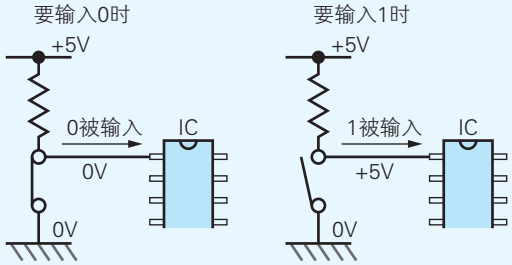

图 2.8　如何用开关输入 0 或 1

连接在 $\overline{\text{RESET}}$ 引脚上的电容，用于在电路接通电源时自动重置 CPU。电容就好像一个充电电池，具有储存电荷的功能。在通电后的一刹那，由于电容正在充电，所以 $\overline{\text{RESET}}$ 引脚上的电压并不会立刻上升到 +5V。而完成充电后，$\overline{\text{RESET}}$ 引脚的电压会变为 +5V，这样就相当于 $\overline{\text{RESET}}$ 引脚上的值从 0 变成了 1，重置了一次 CPU。

总线是连接到 CPU 中数据引脚、地址引脚、控制引脚上的电路的

统称。使用快动开关可以使 Z80 CPU 的 $\overline{\text{BUSRQ}}$ 引脚（即 Bus Request，总线请求）上的值在 0 和 1 之间切换。若将 $\overline{\text{BUSRQ}}$ 引脚的值设为 0，则 Z80 CPU 从电路中隔离。当处于这种隔离状态时，就可以不通过 CPU，手动地向内存写入程序了。像这样不经过 CPU 而直接从外部设备读写内存的行为叫作 DMA（Direct Memory Access，直接存储器访问）。在诸位所使用的个人计算机里，硬盘等设备要读写内存时使用的就是 DMA。

当 Z80 CPU 从电路中隔离后，$\overline{\text{BUSAK}}$ 引脚（即 Bus Acknowledge，响应总线请求）上的值就会变成 0。也就是说，把 $\overline{\text{BUSRQ}}$ 引脚上的值设成 0 以后，还要确认 $\overline{\text{BUSAK}}$ 引脚上的值已经变成了 0，然后才能进行 DMA。请把 $\overline{\text{BUSAK}}$ 引脚分别连接到 4 个 74367 的 G1 和 G2 引脚上。有关 74367 的作用将在后面说明。

Z80 CPU 的其他控制引脚并未使用。所以要把 $\overline{\text{WAIT}}$ 引脚和 $\overline{\text{NMI}}$ 引脚上的值设为 1，即连接到 +5V 上。之所以在连接时加入电阻，是为了便于今后加入开关等元件。请诸位先记住一个词——上拉（Pull-up），指的就是像这样通过加入电阻把元件的引脚和 +5V 连接起来。剩下的 $\overline{\text{HALT}}$ 引脚和 $\overline{\text{ASTB}}$ 引脚什么都不连接。

Z80 PIO 的 PA0～PA7（PA 表示 Port A）以及 PB0～PB7（PB 表示 Port B）用于与外部设备进行输入输出，所以稍后要把它们分别连接到指拨开关和 LED 上。对于剩下的几个引脚可以这样处理：将 IEI 引脚上拉，IEO 引脚、$\overline{\text{ASTB}}$ 引脚、ARDY 引脚、$\overline{\text{BSTB}}$ 引脚和 BRDY 引脚则什么都不连接。

到此为止，Z80 CPU、TC5517、Z80 PIO 以及时钟发生器上要用到的引脚就都接入电路了。这意味着计算机主机系统的功能完成了。作

为总结，表 2.3 汇总了这几块 IC 上引脚的作用以及电信号的输入输出方向（从各个 IC 的角度看）。

用红铅笔尝试布线的诸位觉得怎么样呢？虽然需要连接的电路有点多，但也并不是太复杂吧？其实计算机的工作原理非常简单。CPU 在时钟信号的控制下解释、执行内存中存储的程序，按照程序中的指令从内存或 I/O 中把数据输入到 CPU 中，在 CPU 内部进行运算，再把运算结果输出到内存或 I/O 中。无论是小型的微型计算机，还是高性能的个人计算机，其工作原理都是相同的。

表 2.3　Z80 CPU、TC5517、Z80 PIO 的引脚作用以及输入输出方向

Z80 CPU		
引脚的代号	方向	作用
A0～A15	输出	指定地址
D0～D7	输入输出	输入输出数据
$\overline{\text{MREQ}}$	输出	把输入输出对象设定为内存
$\overline{\text{IORQ}}$	输出	把输入输出对象设定为 I/O
$\overline{\text{RD}}$	输出	输出数据
$\overline{\text{WR}}$	输出	输入数据
$\overline{\text{BUSRQ}}$	输入	接收 DMA 请求
$\overline{\text{BUSAK}}$	输出	响应 DMA 请求
$\overline{\text{M1}}$	输出	用于同步
$\overline{\text{INT}}$	输入	接收中断请求
CLK	输入	接收时钟信号
$\overline{\text{RESET}}$	输入	重置
$\overline{\text{WAIT}}$	输入	（这里未使用）
$\overline{\text{NMI}}$	输入	（这里未使用）
$\overline{\text{HALT}}$	输出	（这里未使用）
$\overline{\text{RFSH}}$	输出	（这里未使用）

（续）

TC5517		
引脚的代号	方向	作用
A0～A10	输入	指定地址
D0～D7	输入输出	输入输出数据
$\overline{CE}$	输入	在电路中激活 IC
$\overline{RD}$	输入	读出数据
$\overline{WE}$	输入	写入数据

Z80 PIO		
引脚的代号	方向	作用
B/$\overline{A}$	输入	选择端口 B 或端口 A
C/$\overline{D}$	输入	选择控制模式或数据模式
D0～D7	输入输出	从 CPU 读取数据或向 CPU 写入数据
$\overline{CE}$	输入	在电路中激活 IC
$\overline{IORQ}$	输入	在电路中激活 IC
$\overline{M1}$	输入	用于同步
$\overline{INT}$	输出	发出中断请求
$\overline{RD}$	输入	选择是读取数据还是写入数据
CLK	输入	接收时钟信号
PA0～PA7	输入输出	从外部设备读取数据或向外部设备写入数据
PB0～PB7	输入输出	从外部设备读取数据或向外部设备写入数据
$\overline{ASTB}$	输入	（这里未使用）
ARDY	输出	（这里未使用）
$\overline{BSTB}$	输入	（这里未使用）
BRDY	输出	（这里未使用）
IEI	输入	（这里未使用）
IEO	输出	（这里未使用）

2.8　连接外部设备，通过 DMA 输入程序

下面我们继续布线，这次将计算机主机系统和外部设备连接起来。我们要使用 2 个指拨开关和 1 个按键开关，向地址总线引脚和数据总线引脚发送电信号，然后通过 DMA 将数据总线上的数据存储到内存。下面我们就先将这些元件连接到电路中。

首先将图 2.1 中右侧最上方的一个指拨开关连接到作为内存的 TC5517 的数据总线引脚 D0～D7 上。再将它下面紧挨着它的指拨开关连接到 TC5517 的地址总线引脚 A0～A7 上。第 3 个指拨开关则通过电阻接到 +5V 上，这样拨动这个指拨开关就可以输入 +5V 或 0V 的信号了。接下来将用于控制内存写入的按键开关连接到 TC5517 的 $\overline{\text{WE}}$ 引脚上。为了写入数据，还要将 TC5517 的 $\overline{\text{RD}}$ 引脚上拉起来，连接到 +5V 上，然后把 $\overline{\text{CE}}$ 引脚连接到 0V 上。把这些元件都连接起来以后，就可以拨动指拨开关，用二进制数设定地址总线引脚和数据总线引脚上的数据了。设定完后按下按键开关，数据就会被写入 TC5517 中。

但是如果这些开关直接连接到了 TC5517 的各个引脚上，在程序执行时，开关的状态就会对电路产生影响。因此要使用 74367，在程序执行时把开关从电路中隔离出来。74367 是一种叫作“三态总线缓冲器”的 IC。在这个 IC 的电路图符号中，有用三角形标志代表的缓冲器，表示使电信号从右向左直接通过。但是，只有在 74367 的 $\overline{\text{G1}}$ 引脚和 $\overline{\text{G2}}$ 引脚同时为 0 的时候，电信号才能通过。而当 $\overline{\text{G1}}$ 引脚和 $\overline{\text{G2}}$ 引脚同时为 1 时，74367 就会与电路隔离。

一旦打开了 Z80 CPU 的 $\overline{\text{BUSRQ}}$ 引脚连接着的开关，就可以通过 $\overline{\text{BUSAK}}$ 引脚输出 0 得知 CPU 进入了 DMA 状态。因此只要把 $\overline{\text{BUSAK}}$ 引脚连接到 4 个 74367 的 $\overline{\text{G1}}$ 引脚和 $\overline{\text{G2}}$ 引脚上，就可以实现通过

DMA 向内存写入数据了。

2.9 连接用于输入输出的外部设备

布线终于快结束了。下面该轮到把指拨开关和 LED 连接到 Z80 PIO 上了。当微型计算机运行起来后，指拨开关可用于从外部输入数据，LED 可用于向外部输出数据。

用于输入数据的指拨开关，要连接到 Z80 PIO 的 PA0～PA7 引脚上。连接时没有使用 74367 是为了在程序运行中可以通过 Z80 PIO 从指拨开关获得输入的数据。

表示输出数据的 LED 要通过电阻连接到 +5V 上。这里的布线方法依据惯例，输入 0V 点亮 LED（如图 2.9 所示）。LED 要通过 7404 这样的 IC 连接到 Z80 PIO 的 PB0～PB7 引脚上。在 7404 的电路图符号中，末端带有一个小圆圈的三角形符号表示反相器，作用是将左侧输入的电信号反转后（即 0 变 1、1 变 0）输出到右侧。通过这样的设计，当 Z80 PIO 的 PB0～PB7 引脚上的值为 0 时 LED 就会熄灭，为 1 时 LED 就会点亮。

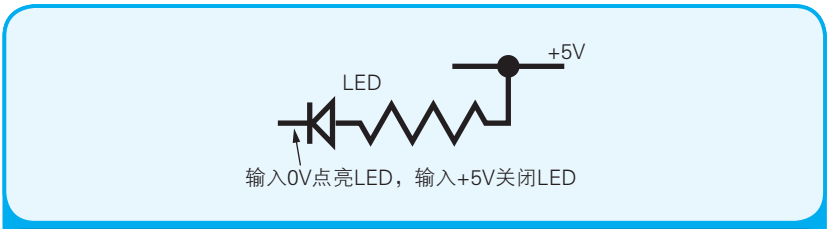

图 2.9 点亮 LED 的方法

是不是觉得忘记了什么呢？没错！ 74367 和 7404 上也都有 Vcc 引脚和 GND 引脚。请将它们分别连接到 +5V 和 0V 上。对于 74367 和

7404 中未使用的引脚（标有 NC 的引脚），或者什么都不连接，或者将它们连接到 GND 上。

2.10　输入测试程序并进行调试

微型计算机终于顺利地制作出来了，诸位辛苦了！微型计算机接上电源就能用了吗？其实还不能，因为尽管硬件组装好了，但若没有输入软件，计算机还是不能工作的。所以即使为微型计算机接通了电源，它也什么都执行不了。

下面就编写一段测试程序吧。编写时可以使用哪种编程语言呢？是 BASIC、C 语言，还是 Java 呢？其实这些语言都无法使用，因为作为计算机大脑的 CPU 只能解释执行一种编程语言，那就是靠罗列二进制数构成的机器语言（原生代码）。代码清单 2.1 展示了一段用机器语言编写的测试程序。程序是指令和数据的集合，表示指令或数据的数值是以 8 比特为一个单位存储到内存中的。这段程序只实现了一个简单的功能，那就是通过拨动连接到 Z80 PIO 上的指拨开关控制 LED 的亮或灭。有关机器语言的细节将在接下来的第 3 章中介绍。

代码清单 2.1　用机器语言编写的测试程序

```
地址            程序
00000000        00111110
00000001        11001111
00000010        11010011
00000011        00000010
00000100        00111110
00000101        11111111
00000110        11010011
00000111        00000010
00001000        00111110
00001001        11001111
00001010        11010011
00001011        00000011
00001100        00111110
```