新 8051 教程——前言

传统的单片机教学，均是以单片机的结构为主线，先讲单片机的硬件结构，然后是指令，然后是软件编程，然后是单片机系统的扩展和各种外围器件的应用，最后再讲一些实例。按照此种教学结构，按照这种结构，学生普遍感到难学。试想，一个从未接触过计算机结构的人，甚至数字电路也是刚刚接触的人，要他去理解单片机内部结构，这实在不是个容易的事，至于很多书一开始就提出的总线、地址等概念，更是初学者难以理解的——不管用什么巧妙的比方都不容易理解。于是糊里糊涂地学完了第一部份，第二部份一开始就是寻址方式，更抽象，好多人直到学完单片机还不能理解寻址方式究竟是什么意思，为什么需要这么多寻址方式，刚开始学当然更不懂了。然后是指令，111 条指令，又不分个重点，反正全是要记住的，等到指令全部学完，大部份人已对单片机望尔生畏，开始打退堂鼓了。第三部份是编程，如果说前面的东西不能理解，还能靠记忆来获得知识的话，这部份就纯是理解和掌握了，如果以前没学过编程，短时间内很难掌握编程的有关知识，更不必说编程技巧了。可是教材上明明规定，要编出这样、那样的程序，学的人编不出来，当然只会认为，教材的要求当然是合理的，应该做到的，我做不到就是我没学好，于是很多人长叹一声：单片机太难学了！放弃吧。可是到这里还根本不知道一个单片机开发的完整过程是什么，什么是编程器还不知道。后面的就不说了，总之，现在教材，基本都是以单片机为蓝本来学习计算机原理，而不纯为学习单片机技术，在教材、教学过程的安排上又没有考虑人的接受能力，使得学习的过程是一个充满不断挫折的过程，于是很多人认为单片机入门难。

基于以上情况，作者尝试编制一套全新的教学方法，以任务为教学单元，打破原有界限，不管硬件结构、指令、编程的先后顺序，将各部份知识分解成一个个知识点，为了完成一个任务抽取每个部份的不同知识点，加以组合，完成第一个任务就能清楚单片机的开发过程，完成第二、三个任务，就能自已模仿性地编出自已的程序，使得学习过程是一个不断成功地完成任务的过程。当所有任务全部完成，知识点就全学完了。即便只完成部份任务，也可以去做一些程序了——事实并没有必要学完全部知识才可以去做开发的，作者在编第一个商用程序时，还不懂定时器怎么用，编第二个商用程序，写了长达 2K 行的代码，可当时我还不懂怎么样用中断编程，因为当时我根本还不需要用中断。

以上的教学方法具有如下特点：

1、以人的认知规律为主线，而不是以课程结构为主线。

2、以任务为单元构建认知单元，而不是以单片机功能为单元构建。

3、完成第一个任务即可进行单片机的初步应用尝试，不必学完单片机的全部知识体系。 随着任务的逐渐进行，知识逐渐完善，能力逐渐提高，所有任务完成时，已具有初步开发能力。

以上的教学目标是一个很'宏伟'的计划，我不知道我最终会不会完成他，因为完成他也许需要二年、三年甚至更长的时间，当然，对于这个计划能否完成，是否合理，我是充满信心的。要完成以上计划，关键在于要做好以下一些工作：知识点的合理分解，合理组合，任务的合理设计等等。我将尽力去做好他，当然，我也希望诸位大虾能不吝赐教，和我共同完

成这个任务，这也算为单片机入门者铺平道路吧。

单片机教程第一课：单片机概述

1、何谓单片机 一台能够工作的计算机要有这样几个部份构成：CPU（进行运算、控制）、RAM（数据存储）、ROM（程序存储）、输入/输出设备（例如：串行口、并行输出口等）。在个人计算机上这些部份被分成若干块芯片，安装一个称之为主板的印刷线路板上。而在单片机中，这些部份，全部被做到一块集成电路芯片中了，所以就称为单片（单芯片）机，而且有一些单片机中除了上述部份外，还集成了其它部份如 A/D，D/A 等。

天！PC 中的 CPU 一块就要卖几千块钱，这么多东西做在一起，还不得买个天价！再说这块芯片也得非常大了。 不，价格并不高，从几元人民币到几十元人民币，体积也不大，一般用 40 脚封装，当然功能多一些单片机也有引脚比较多的，如 68 引脚，功能少的只有 10 多个或 20 多个引脚，有的甚至只 8 只引脚。为什么会这样呢？ 功能有强弱，打个比方，市场上面有的组合音响一套才卖几百块钱，可是有的一台功放机就要卖好几千。另外这种芯片的生产量很大，技术也很成熟，51 系列的单片机已经做了十几年，所以价格就低了。 既然如此，单片机的功能肯定不强，干吗要学它呢？ 话不能这样说，实际工作中并不是任何需要计算机的场合都要求计算机有很高的性能，一个控制电冰箱温度的计算机难道要用 PIII？应用的关键是看是否够用，是否有很好的性能价格比。所以 8051 出来十多年，依然没有被淘汰，还在不断的发展中。

2、MCS51 单片机和 8051、8031、89C51 等的关系我们平常老是讲 8051，又有什么 8031，现在又有 89C51，它们之间究竟是什么关系？MCS51 是指由美国 INTEL 公司（对了，就是大名鼎鼎的 INTEL）生产的一系列单片机的总称，这一系列单片机包括了好些品种，如 8031，8051，8751，8032，8052，8752 等，其中 8051 是最早最典型的产品，该系列其它单片机都是在 8051 的基础上进行功能的增、减、改变而来的，所以人们习惯于用 8051 来称呼 MCS51 系列单片机，而 8031 是前些年在我国最流行的单片机，所以很多场合会看到 8031 的名称。INTEL 公司将 MCS51 的核心技术授权给了很多其它公司，所以有很多公司在做以 8051 为核心的单片机，当然，功能或多或少有些改变，以满足不同的需求，其中 89C51 就是这几年在我国非常流行的单片机，它是由美国 ATMEL 公司开发生产的。以后我们将用 89C51 来完成一系列的实验。

单片机教程第二课：单片机的内部、外部结构(一)

一、单片机的外部结构

拿到一块芯片，想要使用它，首先必须要知道怎样连线，我们用的一块称之为 89C51 的芯片，下面我们就看一下如何给它连线。 1、 电源：这当然是必不可少的了。单片机使用的是 5V 电源，其中正极接 40 引脚，负极（地）接 20 引脚。 2、 振荡电路：单片机是一种时序电路，必须提供脉冲信号才能正常工作，在单片机内部已集成了振荡器，使用晶体振荡器，接 18、19 脚。只要买来晶振，电容，连上就可以了，按图 1 接上即可。 3、 复位引脚：按图 1 中画法连好，至于复位是何含义及为何需要复要复位，在单片机功能中介绍。 4、 EA 引脚：EA 引脚接到正电源端。 至此，一个单片机就接好，通上电，单片机就开始工作了。

我们的第一个任务是要用单片机点亮一只发光二极管 LED，显然，这个 LED 必须要和单片机的某个引脚相连，否则单片机就没法控制它了，那么和哪个引脚相连呢？单片机上除了刚才用掉的 5 个引脚，还有 35 个，我们将这个 LED 和 1 脚相连。（见图 1，其中 R1 是限流电阻）

按照这个图的接法，当 1 脚是高电平时，LED 不亮，只有 1 脚是低电平时，LED 才发亮。因此要 1 脚我们要能够控制，也就是说，我们要能够让 1 引脚按要求变为高或低电平。即然我们要控制 1 脚，就得给它起个名字，总不能就叫它一脚吧？叫它什么名字呢？设计 51 芯片的 INTEL 公司已经起好了，就叫它 P1.0，这是规定，不可以由我们来更改。

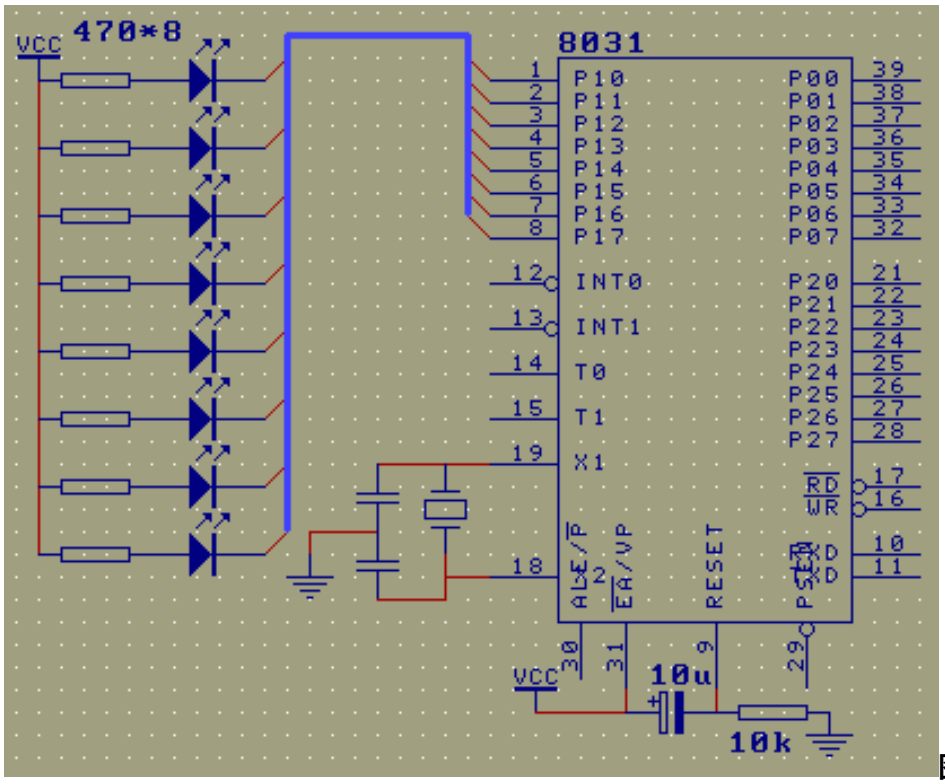


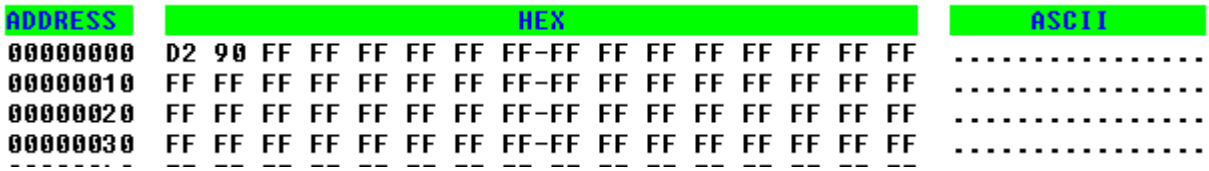
图 1

名字有了，我们又怎样让它变'高'或变'低'呢？叫人做事，说一声就可以，这叫发布命令，要

计算机做事，也得要向计算机发命令，计算机能听得懂的命令称之为计算机的指令。让一个引脚输出高电平的指令是 SETB，让一个引脚输出低电平的指令是 CLR。因此，我们要 P1.0 输出高电平，只要写 SETB P1.0，要 P1.0 输出低电平，只要写 CLR P1.0 就可以了。

现在我们已经有办法让计算机去将 P10 输出高或低电平了，但是我们怎样才能计算机执行这条指令呢？总不能也对计算机也说一声了事吧。要解决这个问题，还得有几步要走。第一，计算机看不懂 SETB CLR 之类的指令，我们得把指令翻译成计算机能懂的方式，再让计算机去读。计算机能懂什么呢？它只懂一样东西 —数字。因此我们得把 SETB P1.0 变为（D2H,90H ），把 CLR P1.0 变为 （C2H,90H ），至于为什么是这两个数字，这也是由 51 芯片的设计者--INTEL 规定的，我们不去研究。第二步，在得到这两个数字后，怎样让这两个数字进入单片机的内部呢？这要借助于一个硬件工具"编程器"。

我们将编程器与电脑连好，运行编程器的软件，然后在编缉区内写入（D2H,90H）见图 2，写入 …..好，拿下片子，把片子插入做好的电路板，接通电源 …..什么?灯不亮？这就对了，因为我们写进去的指令就是让



ADDRESS	HEX	ASCII
00000000	D2 90 FF FF FF FF FF FF-FF FF FF FF FF FF FF FF	....
00000010	FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF	....
00000020	FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF	....
00000030	FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF	....

图 2

P10 输出高电平，灯当然不亮，要是亮就错了。现在我们再拨下这块芯片，重新放回到编程器上，将编缉区的内容改为（C2H,90H），也就是 CLR P1.0，写片，拿下片子，把片子插进电路板，接电，好，灯亮了。因为我们写入的（）就是让 P10 输出低电平的指令。这样我们看到，硬件电路的连线没有做任何改变，只要改变写入单片机中的内容，就可以改变电路的输出效果。

三、单片机内部结构分析

我们来思考一个问题，当我们在编程器中把一条指令写进单片要内部，然后取下单片机，单片机就可以执行这条指令，那么这条指令一定保存在单片机的某个地方，并且这个地方在单片机掉电后依然可以保持这条指令不会丢失，这是个什么地方呢？这个地方就是单片机内部的只读存储器即 ROM（READ ONLY MEMORY）。为什么称它为只读存储器呢？刚才我们不是明明把两个数字写进去了吗？原来在 89C51 中的 ROM 是一种电可擦除的 ROM，称为 FLASH ROM，刚才我们是用的编程器，在特殊的条件下由外部设备对 ROM 进行写的操作，在单片机正常工作条件下，只能从那面读，不能把数据写进去，所以我们还是把它称为 ROM。

单片机教程第三课：几个基本概念

数的本质和物理现象。

我们知道，计算机可以进行数学运算，这可令我们非常的难以理解，计算机吗，我们虽不了解它的组成，但它总只是一些电子元器件，怎么可以进行数学运算呢？我们做数学题如 37+45 是这样做的，先在纸上写 37，然后在下面写 45，然后大脑运算，最后写出结果，运算的原材料：37、45 和结果：82 都是写在纸上的，计算机中又是放在什么地方呢？为了解决这个问题，先让我们做一个实验：这里有一盏灯，我们知道灯要么亮，要么不亮，就有两种状态，我们可以用'0'和'1'来代替这两种状态，规定亮为'1'，不亮为'0'。现在放上两盏灯，一共有几种状态呢？我们列表来看一下：

状态				
表达	0 0	0 1	1 0	1 1

请大家自已写上 3 盏灯的情况 000 001 010 011 100 101 110 111

我们来看，这个 000，001，101 不就是我们学过的的二进制数吗？本来，灯的亮和灭只是一种物理现象，可当我们把它们按一按的顺序排更好后，灯的亮和灭就代表了数字了。让我们再抽象一步，灯为什么会亮呢？看电路 1，是因为输出电路输出高电平，给灯通了电。因此，灯亮和灭就可以用电路的输出是高电平还是低电平来替代了。这样，数字就和电平的高、低联系上了。（请想一下，我们还看到过什么样的类似的例子呢？（海军之）灯语、旗语，电报，甚至红、绿灯）

位的含义： 通过上面的实验我们已经知道：一盏灯亮或者说一根线的电平的高低，可以代表两种状态：0 和 1。实际上这就是一个二进制位，因此我们就把一根线称之为一“位”，用 BIT 表示。

字节的含义： 一根线可以表于 0 和 1，两根线可以表达 00，01，10，11 四种状态，也就是可以表于 0 到 3，而三根可以表达 0-7，计算机中通常用 8 根线放在一起，同时计数，就可以表过到 0-255 一共 256 种状态。这 8 根线或者 8 位就称之为一个字节（BYTE）。不要问我为什么是 8 根而不是其它数，因为我也不知道。（计算机世界是一个人造的世界，不是自然界，很多事情你无法问为什么，只能说：它是一种规定，大家在以后的学习过程中也要注意这个问题）

存储器的工作原理：

1、存储器构造

存储器就是用来存放数据的地方。它是利用电平的高低来存放数据的，也就是说，它存放的实际上是电平的高、低，而不是我们所习惯认为的 1234 这样的数字，这样，我们的一个谜团就解开了，计算机也没什么神秘的吗。

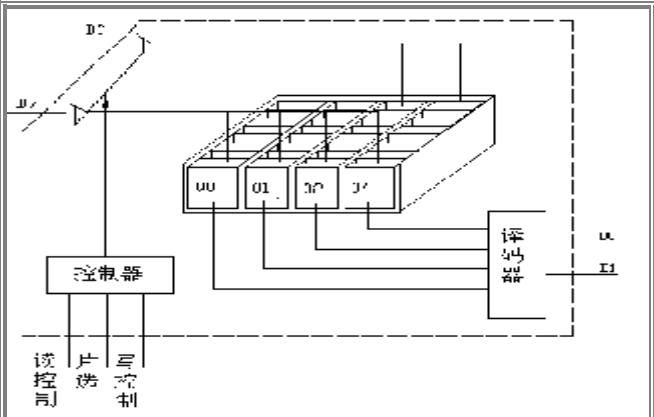


图 2

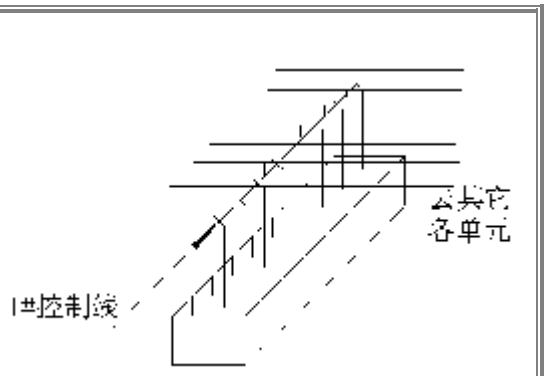


图 3

让我们看图 2。这是一个存储器的示意图：一个存储器就象一个个的小抽屉，一个小抽屉里有八个小格子，每个小格子就是用来存放“电荷”的，电荷通过与它相连的电线传进来或释放掉，至于电荷在小格子里是怎样存的，就不用我们操心了，你可以把电线想象成水管，小格子里的电荷就象是水，那就好理解了。存储器中的每个小抽屉就是一个放数据的地方，我们称之为一个“单元”。

有了这么一个构造，我们就可以开始存放数据了，想要放进一个数据 12，也就是 00001100，我们只要把第二号和第三号小格子里存满电荷，而其它小格子里的电荷给放掉就行了（看图 3）。可是问题出来了，看图 2，一个存储器有好多单元，线是并联的，在放入电荷的时候，会将电荷放入所有的单元中，而释放电荷的时候，会把每个单元中的电荷都放掉，这样的话，不管存储器有多少个单元，都只能放同一个数，这当然不是我们所希望的，因此，要在结构上稍作变化，看图 2，在每个单元上有个控制线，我想要把数据放进哪个单元，就给一个信号这个单元的控制线，这个控制线就把开关打开，这样电荷就可以自由流动了，而其它单元控制线上没有信号，所以开关不打开，不会受到影响，这样，只要控制不同单元的控制线，就可以向各单元写入不同的数据了，同样，如果要某个单元中取数据，也只要打开相应的控制开关就行了。

2、存储器译码

那么，我们怎样来控制各个单元的控制线呢？这个还不简单，把每个单元元的控制线都引到集成电路的外面不就行了吗？事情可没那么简单，一片 27512 存储器中有 65536 个单元，把每根线都引出来，这个集成电路就得有 6 万多个脚？不行，怎么办？要想法减少线的数量。我们有一种方法称这为译码，简单介绍一下：一根线可以代表 2 种状态，2 根线可以代表 4 种状态，3 根线可以代表几种，256 种状态又需要几根线代表？8 种，8 根线，所以 65536 种状态我们只需要 16 根线就可以代表了。

3、存储器的选片及总线的概念

至此，译码的问题解决了，让我们再来关注另外一个问题。送入每个单元的八根线是用从什么地方来的呢？它就是从计算机上接过来的，一般地，这八根线除了接一个存储器之外，还要接其它的器件，如图 4 所示。这样问题就出来了，这八根线既然不是存储器和计算机之间专用的，如果总是将某个单元接在这八根线上，就不好了，比如这个存储器单元中的数值

是 0FFH 另一个存储器的单元是 00H，那么这根线到底是处于高电平，还是低电平？岂非要打架看谁历害了？所以我们要让它们分离。办法当然很简单，当外面的线接到集成电路的引脚进来后，不直接接到各单元去，中间再加一组开关（参考图 4）就行了。平时我们让开关打开着，如果确实是要向这个存储器中写入数据，或要从存储器中读出数据，再让开关接通就行了。这组开关由三根引线选择：读控制端、写控制端和片选端。要将数据写入片中，先选中该片，然后发出写信号，开关就合上了，并将传过来的数据（电荷）写入片中。如果要读，先选中该片，然后发出读信号，开关合上，数据就被送出去了。注意图 4，读和写信号同时还接入到另一个存储器，但是由于片选端不同，所以虽有读或写信号，但没有片选信号，所以另一个存储器不会“误会”而开门，造成冲突。那么会不同时选中两片芯片呢？只要是设计好的系统就不会，因为它是由计算控制的，而不是我们人来控制的，如果真的出现同时出现选中两片的情况，那就是电路出了故障了，这不在我们的讨论之列。

从上面的介绍中我们已经看到，用来传递数据的八根线并不是专用的，而是很多器件大家共用的，所以我们称之为数据总线，总线英文名为 BUS，即公交车道，谁者可以走。而十六根地址线也是连在一起的，称之为地址总线。

半导体存储器的分类

按功能可以分为只读和随机存取存储器两大类。所谓只读，从字面上理解就是只可以从里面读，不能写进去，它类似于我们的书本，发到我们手回之后，我们只能读里面的内容，不可以随意更改书本上的内容。只读存储器的英文缩写为 ROM（READ ONLY MEMORY）

所谓随机存取存储器，即随时可以改写，也可以读出里面的数据，它类似于我们的黑板，我可以随时写东西上去，也可以用黑板擦擦掉重写。随机存储器的英文缩写为 RAM（READ RANDOM MEMORY）这两种存储器的英文缩写一定要记牢。

注意：所谓的只读和随机存取都是指在正常工作情况下而言，也就是在使用这块存储器的时候，而不是指制造这块芯片的时候。否则，只读存储器中的数据是怎么来的呢？其实这个道理也很好理解，书本拿到我们手里是不能改了，可以当它还是原材料——白纸的时候，当然可以由印刷厂印上去了。

顺便解释一下其它几个常见的概念。

PROM，称之为可编程存储器。这就象我们的练习本，买来的时候是空白的，可以写东西上去，可一旦写上去，就擦不掉了，所以它只能用写一次，要是写错了，就报销了。

EPROM，称之为紫外线擦除的可编程只读存储器。它里面的内容写上去之后，如果觉得不满意，可以用一种特殊的方法去掉后重写，这就是用紫外线照射，紫外线就象“消字灵”，可以把字去掉，然后再重写。当然消的次数多了，也就不灵光了，所以这种芯片可以擦除的次数也是有限的——几百次吧。

FLASH，称之为闪速存储器，它和 EPROM 类似，写上去的东西也可以擦掉重写，但它要方便一些，不需要光照了，只要用电学方法就可以擦除，所以就方便许多，而且寿面也很长（几万到几十万次不等）。

再次强调，这里的所有的写都不是指在正常工作条件下。不管是 PROM、EPROM 还是 FLASH ROM，它们的写都要有特殊的条件，一般我们用一种称之为“编程器”的设备来做这项工作，一旦把它装到它的工作位置，就不能随便改写了。

单片机教程第四课：第一个小程序

上一次我们的程序实在是没什么用，要灯亮还要重写一下片子，下面我们要让灯不断地闪烁，这就有一定的实用价值了，比如可以把它当成汽车上的一个信号灯用了。怎样才能让灯不断地闪烁呢？实际上就是要灯亮一段时间，再灭一段时间，也就是说要 P10 不断地输出高和低电平。怎样实现这个要求呢？请考虑用下面的指令是否可行：

```
SETB P10
CLR P10 .….
```

这是不行的，有两个问题，第一，计算机执行指令的时间很快，执行完 SETB P10 后，灯是灭了，但在极短时间（微秒级）后，计算机又执行了 CLR P10 指令，灯又亮了，所以根本分辨不出灯曾灭过。第二，在执行完 CLR P10 后，不会再去执行 SETB P10 指令，所以以后再也没有机会让灭了。

为了解决这两个问题，我们可以做如下设想，第一，在执行完 SETB P10 后，延时一段时间（几秒或零点几秒）再执行第二条指令，就可以分辨出灯曾灭过了。第二在执行完第二条指令后，让计算机再去执行第一条指令，不断地在原地兜圈，我们称之为"循环"，这样就可以完成任务了。

以下先给出程序（后面括号中的数字是为了便于讲解而写的，实际不用输入）：

```
；主程序：
LOOP：  SETB P10          ；（1）
        LCALL DELAY       ；（2）
        CLR P10           ；（3）
        LCALL DELAY       ；（4）
        AJMP LOOP         ；（5）
；以下子程序
DELAY：  MOV R7，#250     ；（6）
D1：  MOV R6，#250        ；（7）
D2：  DJNZ R6，D2         ；（8）
      DJNZ R7，D1         ；（9）
      RET                 ；（10）
      END                 ；（11）
```

按上面的设想分析一下前面的五条指令。

第一条是让灯灭，第二条应当是延时，第三条是让灯亮，第四条和第二条一模一样，也是延时，第五条应当是转去执行第一条指令。第二和第四条实现的原理稍后谈，先看第五条，LJMP 是一条指令，意思是转移，往什么地方转移呢？后面跟的是 LOOP，看一下，什么地方还有 LOOP，对了，在第一条指令的前面有一个 LOOP，所以很直观地，我们可以认识到，它要转到第一条指令处。这个第一条指令前面的 LOOP 被称之为标号，它的用途就是给这一行起一个名字，便于使用。是否一定要给它起名叫 LOOP 呢？当然不是，起什么名字，完全由编程序的人决定，可以称它为 A，X 等等，当然，这时，第五条指令 LJMP 后面的名字也得跟着改了。

第二条和第四条指令的用途是延时，它是怎样实现的呢？指令的形式是 LCALL，这条指令称为调用子程序指令，看一下指令后面跟的是什么，DELAY，找一下 DELAY，在第六条指令的前面，显然，这也是一个标号。这条指令的作用是这样的：当执行 LCALL 指令时，

程序就转到 LCALL 后面的标号所标定的程序处执行，如果在执行指令的过程中遇到 RET 指令，则程序就返回到 LCALL 指令的下面的一条指令继续执行，从第六行开始的指令中，可以看到确实有 RET 指令。在执行第二条指令后，将转去执行第 6 条指令，而在执行完 6，7，8，9 条指令后将遇到第 1 0 条令：RET，执行该条指令后，程序将回来执行第三条指令，即将 P10 清零，使灯亮，然后又是第四条指令，执行第四条指令就是转去执行第 6，7，8，9，10 条指令，然后回来执行第 5 条指令，第 5 条指令就是让程序回到第 1 条开始执行，如此周而复始，灯就在不断地亮、灭了。

在标号 DELAY 标志的这一行到 RET 这一行中的所有程序，这是一段延时程序，大概延时零点几秒，至于具体的时间，以后我们再学习如何计算。 程序的最后一行是 END，这不是一条指令，它只是告诉我们程序到此结束，它被称为"伪指令"。

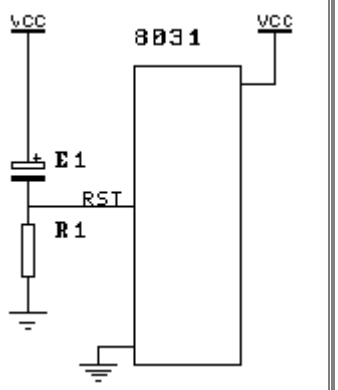
单片机内部结构分析

为了知道延时程序是如何工作的，我们必需首先了解延时程序中出现的一些符号， 就 R1 开始，R1 被称之为工作寄存器。什么是工作寄存器呢？让我们从现实生活中来找找答案。如果出一道数学题：123+567，让你回答结果是多少，你会马上答出是 690，再看下面一道题：123+567+562，要让你要上回答，就不这么容易了吧？我们会怎样做呢？如果有张纸，就容易了，我们先算出 123+567=690，把 690 写在纸上，然后再算 690+562 得到结果是 1552。这其中 1552 是我们想要的结果，而 690 并非我们所要的结果，但是为了得到最终结果，我们又不得不先算出 690，并记下来，这其实是一个中间结果，计算机中做运算和这个类似，为了要得到最终结果，往往要做很多步的中间结果，这些中间结果要有个地方放才行，把它们放哪呢？放在前面提到过的 ROM 中可以吗？显然不行，因为计算机要将结果写进去，而 ROM 是不可以写的，所以在单片机中另有一个区域称为 RAM 区（RAM 是随机存取存储器的英文缩写），它可以将数据写进去。 特别地，在 MCS-51 单片机中，将 RAM 中分出一块区域，称为工作寄存器区

单片机教程第五课：延时程序分析

上一次课中，我们已经知道，程序中的符号 R7、R6 **是代表了一个个的** RAM **单元，是用来放一些数据的，下面我们再来看一下其它符号的含义。**

```
DELAY：  MOV R7，#250      ；（ 6 ）
D1：  MOV R6，#250        ；（ 7 ）
D2：  DJNZ R6，D2         ；（ 8 ）
DJNZ R7，D1          ；（ 9 ）
RET                  ；（ 1 0 ）
```



MOV：**这是一条指令，意思是传递数据。说到传递，我们都很清楚，传东西要从一个人的手上传到另一个人的手上，也就是说要有一个接受者，一个传递者和一样东西。从指令** MOV R7，#250 **中来分析，** R7 **是一个接受者，** 250 **是被传递的数，传递者在这条指令中被省略了（注意：并不是每一条传递指令都会省的，事实上大部份数据传递指令都会有传递者）。它的意义也很明显：将数据** 250 **送到** R7 **中去，因此执行完这条指令后，** R7 **单元中的值就应当是** 250**。在** 250 **前面有个#号，这又是什么意思呢？这个#就是用来说明** 250 **就是一个被传递的东西本身，而不是传递者。那么** MOV R6，#250 **是什么意思，应当不用分析了吧。**

DJNZ：**这是另一条指令，我们来看一下这条指令后面跟着的两个东西，一个是** R6，**一个是** D2，R6 **我们当然已知是什么了，查一下** D2 **是什么。**D2 **在本行的前面，我们已学过，这称之为标号。标号的用途是什么呢？就是给本行起一个名字。**DJNZ **指令的执行过程是这样的，它将其后面的第一个参数中的值减** 1，**然后看一下，这个值是否等于** 0，**如果等于** 0，**就往下执行，如果不等于** 0，**就转移，转到什么地方去呢？可能大家已猜到了，转到第二个参数所指定的地方去（请大家用自已的话讲一下这条语句是怎样执行的）。本条指令的最终执行结果就是，在原地转圈** 250 **次。**

执行完了 DJNZ R6，D2 **之后（也就是** R6 **的值等于** 0 **之后），就会去执行下面一行，也就是** DJNZ R7，D1，**请大家自行分析一下这句话执行的结果。（转去执行** MOV R6，#250，**同时** R7 **中的值减** 1），**最终** DJNZ R6，D2 **这句话将被执行** 250*250=62500 **次，执行这么多次同一条指令干吗？就是为了延时。**

一个问题：如果在 R6 **中放入** 0，**会有什么样的结果。**

二、时序分析：

前面我们介绍了延时程序，但这还不完善，因为，我们只知道 DJNZ R6，D2 **这句话会被执行** 62500 **次，但是执行这么多次需要多长时间呢？是否满足我们的要求呢？我们还不知道，所以下面要来解决这个问题。**

先提一个问题：我们学校里什么是最重要的。（铃声）校长可以出差，老师可以休息，但学校一日无铃声必定大乱。整个学校就是在铃声的统一指挥下，步调一致，统一协调地工作着。这个铃是按一定的时间安排来响的，我们可以称之为“时序&#0;&#0;时间的顺序”。一个由人组成的单位尚且要有一定的时序，计算机当然更要有严格的时序。事实上，计算机更象一个大钟，什么时候分针动，什么时候秒针动，什么时候时针动，都有严格的规定，一点也不能乱。计算机要完成的事更复杂，所以它的时序也更复杂。

我们已知，计算机工作时，是一条一条地从 ROM 中取指令，然后一步一步地执行，我们规定：计算机访问一次存储器的时间，称之为一个机器周期。这是一个时间基准，好象我们人用“秒”作为我们的时间基准一样，为什么不干脆用“秒”，多好，很习惯，学下去我们就会知道用“秒”反而不习惯。

一个机器周期包括 12 个时钟周期。下面让我们算一下一个机器周期是多长时间吧。设一个单片机工作于 12M 晶振，它的时钟周期是 1/12（微秒）。它的一个机器周期是 12*（1/12）也就是 1 微秒。（请计算一个工作于 6M 晶振的单片机，它的机器周期是多少）。

MCS-51 单片机的所有指令中，有一些完成得比较快，只要一个机器周期就行了，有一些完成得比较慢，得要 2 个机器周期，还有两条指令要 4 个机器周期才行。这也不难再解，不是吗？我让你扫地的执行要完成总得比要你完成擦黑板的指令时间要长。为了恒量指令执行时间的长短，又引入一个新的概念：指令周期。所谓指令周期就是指执行一条指令的时间。INTEL 对每一条指令都给出了它的指令周期数，这些数据，大部份不需要我们去记忆，但是有一些指令是需要记住的，如 DJNZ 指令是双周期指令。

下面让我们来计算刚才的延时。首先必须要知道晶振的频率，我们设所用晶振为 12M，则一个机器周期就是 1 微秒。而 DJNZ 指令是双周期指令，所以执行一次要 2 个微秒。一共执行 62500 次，正好 125000 微秒，也就是 125 毫秒。

练习：设计一个延时 100 毫秒的延时程序。

要点分析：1、一个单元中的数是否可以超过 255。2、如何分配两个数。

三、复位电路

任何单片机在工作之前都要有个复位的过程，复位是什么意思呢？它就象是我们上课之前打的预备铃。预备铃一响，大家就自动地从操场、其它地方进入教室了，在这一段时间里，是没有老师干预的，对单片机来说，是程序还没有开始执行，是在做准备工作。显然，准备工作不需要太长的时间，复位只需要 5ms 的时间就可以了。如何进行复位呢？只要在单片机的 RST 引脚上加上高电平，就可以了，按上面所说，时间不少于 5ms。为了达到这个要求，可以用很多种方法，这里提供一种供参考，见图 1。实际上，我们在上一次实验的图中已见到过了。

这种复位电路的工作原理是：通电时，电容两端相当于是短路，于是 RST 引脚上为高电平，然后电源通过电阻对电容充电，RST 端电压慢慢下降，降到一定程序，即为低电平，单片机开始正常工作。

单片机教程第六课：单片机的内外部结构分析（四）

上两次我们做过两个实验，都是让 P1.0 这个引脚使灯亮，我们可以设想：既然 P1.0 可以让灯亮，那么其它的引脚可不可以呢？看一下图 1，它是 8031 单片机引脚的说明，在 P1.0 旁边有 P1.1，P1.2 …P1.7，它们是否都可以让灯亮呢？除了以 P1 开头的外，还有以 P0，P2，P3 开头的，数一下，一共是 32 个引脚，前面我们以学过 7 个引脚，加上这 32 个这 39 个了。它们都以 P 字开头，只是后面的数字不一样，它们是否有什么联系呢？它们能不能都让灯亮呢？在我们的实验板上，除了 P10 之外，还有 P11&#0;&#0;P17 都与 LED 相连，下面让我们来做一个实验，程序如下：

```
MAIN： MOV P1，#0FFH
LCALL DELAY
MOV P1，#00H
LCALL DELAY
LJMP MAIN
DELAY：MOV R7，#250
D1： MOV R6，#250
D2： DJNZ R6，D2
DJNZ R7，D1
RET
END
```

将这段程序转为机器码，用编程器写入芯片中，结果如何？通电以后我们可以看到 8 只 LED 全部在闪动。因此，P10~P17 是全部可以点亮灯的。事实上，凡以 P 开头的这 32 个引脚都是可以点亮灯的，也就是说：这 32 个引脚都可以作为输出使用，如果不用来点亮 LED，可以用来控制继电器，可以用来控制其它的执行机构。

程序分析：这段程序和前面做过的程序比较，只有两处不一样：第一句：原来是 SETB P1.0，现在改为 MOV P1，#0FFH，第三句：原来是 CLR P1.0，现在改为 MOV P1.0，#00H。从中可以看出，P1 是 P1.0&#0;&#0;P1.7 的全体的代表，一个 P1 就表示了所有的这八个管脚了。当然用的指令也不一样了，是用 MOV 指令。为什么用这条指令？看图 2，我们把 P1 作为一个整体，就把它当作是一个存储器的单元，对一个单元送进一个数可以用 MOV 指令。

二、第四个实验

除了可以作为输出外，这 32 个引脚还可以做什么呢？下面再来做一个实验，程序如下：

```
MAIN： MOV P3，#0FFH
LOOP： MOV A，P3
MOV P1，A
LJMP LOOP
```

先看一下实验的结果：所有灯全部不亮，然后我按下一个按钮，第（）个灯亮了，再按下另一个按钮，第（）个灯亮了，松开按钮灯就灭了。从这个实验现象结合电路来分析一下程序。

从硬件电路的连线可以看出，有四个按钮被接入到 P3 口的 P32，P33，P34，P35。第一条指

令的用途我们可以猜到：使 P3 口全部为高电平。第二条指令是 MOV A，P3，其中 MOV 已经见，是送数的意思，这条指令的意思就是将 P3 口的数送到 A 中去，我们可以把 A 当成是一个中间单元（看图 3），第三句话是将 A 中的数又送到 P1 口去，第四句话是循环，就是不断地重复这个过程，这我们已见过。当我们按下第一个按钮时，第（3）只灯亮了，所以 P12 口应当输出是低电平，为什么 P12 口会输出低电平呢？我们看一下有什么被送到了 P1 口，只有从 P3 口进来的数送到 A，又被送到了 P1 口，所以，肯定是 P3 口进来的数使得 P12 位输出电平的。P3 口的 P32 位的按钮被按下，使得 P32 位的电平为低，通过程序，又使 P12 口输出低电平，所以 P3 口起来了一个输入的作用。验证：按第二、三、四个按钮，同时按下 2 个、3 个、4 个按钮都可以得到同样的结论，所以 P3 口确实起到了输入作用，这样，我们可以看到，以 P 字开头的管脚，不仅可以用作输出，还可以用作输入，其它的管脚是否可以呢？是的，都可以。这 32 个引脚就称之为并行口，下面我们就对并行口的结构作一个分析，看一下它是怎样实现输入和输出的。

并行口结构分析：

1、输出结构

先看 P1 口的一位的结构示意图（只画出了输出部份）：从图中可以看出，开关的打开和合上代表了引脚输出的高和低，如果开关合上了，则引脚输出就是低，如果开关打开了，则输出高电平，这个开关是由一根线来控制的，这根数据总线是出自于 CPU，让我们回想一下，数据总线是一根大家公用的线，很多的器件和它连在一起，在不同的时候，不同的器件当然需要不同的信号，如某一时刻我们让这个引脚输出高电平，并要求保持若干时间，在这段时间里，计算机当然在忙个不停，在与其它器件进行联络，这根控制线上的电平未必能保持原来的值不变，输出就会发生变化了。怎么解决这个问题呢？我们在存储器一节中学过，存储器中是可以存放电荷的，我们不妨也加一个小的存储器的单元，并在它的前面加一个开关，要让这一位输出时，就把开关打开，信号就进入存储器的单元，然后马上关闭开关，这样这一位的状态就被保存下来，直到下一次命令让它把开关再打开为止。这样就能使这一位的状态与别的器件无关了，这么一个小单元，我们给它一个很形象的名字，称之为“锁存器”。

2、输入结构

这是并行口的一位的输出结构示意图，再看，除了输出之外，还有两根线，一根从外部引脚接入，另一根从锁存器的输出接出，分别标明读引脚和读锁存器。这两根线是用于从外部接收信号的，为什么要两根呢？原来，在 51 单片机中输入有两种方式，分别称为‘读引脚’和‘读锁存器’，第一种方式是将引脚作为输入，那是真正地从外部引脚读进输入的值，第二种方式是该引脚处于输出状态时，有时需要改变这一位的状态，则并不需要真正地读引脚状态，而只是读入锁存器的状态，然后作某种变换后再输出。

请注意输入结构图，如果将这一根引线作为输入口使用，我们并不能保证在任何时刻都能得到正确的结果（为什么？）参考图 2 输入示意图。接在外部的开关如果打开，则应当是输入 1，而如果闭合开关，则输入 0，但是，如果单片机内部的开关是闭合的，那么不管外部的开关是开还是闭，单片机接受到的数据都是 0。可见，要让这一端口作为输入使用，要先做一个‘准备工作’，就是先让内部的开关断开，也就是让端口输出‘1’才行。正因为要先做这么一个准备工作，所以我们称之为“准双向 I/O 口”。

以上是 P1 口的一位的结构，P1 口其它各位的结构与之相同，而其它三个口：P0、P2、

P3 则除入作为输入输出口之外还有其它用途，所以结构要稍复杂一些，但其用于输入、输出的结构是相同的。看图（）。对我们来说，这些附加的功能不必由我们来控制，所以我们就不去关心它了。

单片机教程第七课：单片机内部结构分析（五）

通过前面的学习，我们已知单片机的内部有 ROM、有 RAM、有并行 I/O 口，那么，除了这些东西之外，单片机内部究竟还有些什么，这些个零碎的东西怎么连在一起的，让我们来对单片机内部作一个完整的分析吧！

看图（1）（本图太大，请大家找本书看吧，一般讲单片机的书，随便哪本都有）。从图中我们可以看出，在 51 单片机内部有一个 CPU 用来运算、控制，有四个并行 I/O 口，分别是 P0、P1、P2、P3，有 ROM，用来存放程序，有 RAM，用来存放中间结果，此外还有定时/计数器，串行 I/O 口，中断系统，以及一个内部的时钟电路。在一个 51 单片机的内部包含了这么多的东西。

对上面的图进行进一步的分析，我们已知，对并行 I/O 口的读写只要将数据送入到相应 I/O 口的锁存器就可以了，那么对于定时/计数器，串行 I/O 口等怎么用呢？在单片机中有一些独立的存储单元是用来控制这些器件的，被称之为特殊功能寄存器（SFR）。事实上，我们已接触过 P1 这个特殊功能寄存器了，还有哪些呢？看表 1

符号	地址	功能介绍
B	F0H	B 寄存器
ACC	E0H	累加器
PSW	D0H	程序状态字
IP	B8H	中断优先级控制寄存器
P3	B0H	P3 口锁存器
IE	A8H	中断允许控制寄存器
P2	A0H	P2 口锁存器
SBUF	99H	串行口锁存器
SCON	98H	串行口控制寄存器
P1	90H	P1 口锁存器
TH1	8DH	定时器/计数器 1（高 8 位）
TH0	8CH	定时器/计数器 1（低 8 位）
TL1	8BH	定时器/计数器 0（高 8 位）
TL0	8AH	定时器/计数器 0（低 8 位）
TMOD	89A	定时器/计数器方式控制寄存器
TCON	88H	定时器/计数器控制寄存器
DPH	83H	数据地址指针（高 8 位）
DPL	82H	数据地址指针（低 8 位）
SP	81H	堆栈指针
P0	80H	P0 口锁存器
PCON	87H	电源控制寄存器

表 1

下面，我们介绍一下几个常用的 SFR，看图 2。
ACC：累加器，通常用 A 表示。这是个什么东西，可不能从名字上理解，它是一个寄存器，而不是一个做加法的东西，为什么给它这么一个名字呢？或许是因为在运算器做运算时其中一个数一定是在 ACC 中的缘故吧。它的名字特殊，身份也特殊，稍后我们将学到指令，可以发现，所有的运算类指令都离不开它。
2、B：一个寄存器。在做乘、除法时放乘数或除数，不做乘除法时，随你怎么用。
3、PSW：程序状态字。这是一个很重要的东西，里面放了 CPU 工作时的很多状态，借此，我们可以了解 CPU 的当前状态，并作出相应的处理。它的各位功能请看表 2

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV		P

表 2
下面我们逐一介绍各位的用途
（1）CY：进位标志。8051 中的运算器是一种 8 位的运算器，我们知道，8 位运算器只能表示到 0-255，如果做加法的话，两数相加可能会超过 255，这样最高位就会丢失，造成运算的错误，怎么办？最高位就进到这里来。这样就没事了。
例：78H+97H（01111000+10010111）
（2）AC：半进位标志。
例：57H+3AH（01010111+00111010）
（3）F0：用户标志位，由我们（编程人员）决定什么时候用，什么时候不用。
（4）RS1、RS0：工作寄存器组选择位。这个我们已知了。
（5）0V：溢出标志位。什么是溢出我们稍后再谈吧。
（6）P：奇偶校验位：它用来表示 ALU 运算结果中二进制数位“1”的个数的奇偶性。若为奇数，则 P=1，否则为 0。
例：某运算结果是 78H（01111000），显然 1 的个数为偶数，所以 P=0。

4、DPTR（DPH、DPL）：

数据指针，可以用它来访问外部数据存储器中的任一单元，如果不用，也可以作为通用寄存器来用，由我们自己决定如何使用。

5、P0、P1、P2、P3：

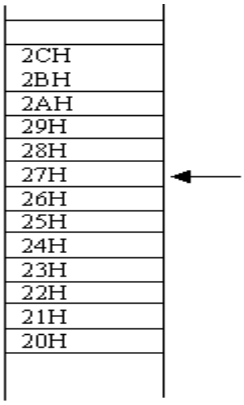
这个我们已经知道，是四个并行输入/输出口的寄存器。它里面的内容对应着管脚的输出。

6、SP：堆栈指针。

堆栈介绍：日常生活中，我们都注意到过这样的现象，家里洗的碗，一只一只摞起来，最晚放上去的放在最上面，而最早放上去的则放在最下面，在取的时候正好相反，先从最上面取，这种现象我们用一句话来概括：“先进后出，后进先出”。请大家想想，还有什么地方有这种现象？其实比比皆是，建筑工地上堆放的砖头、材料，仓库里放的货物，都是“先进后出，后进先出”，这实际是一种存取物品的规则，我们称之为“堆栈”。
在单片机中，我们也可以在 RAM 中构造这样一个区域，用来存放数据，这个区域存放数据的规则就是“先进后出，后进先出”，我们称之为“堆栈”。为什么需要这样来存放数据呢？

存储器本身不是可以按地址来存放数据吗？对，知道了地址的确就可以知道里面的内容，但如果我们需要存放的是一批数据，每一个数据都需要知道地址那不是麻烦吗？如果我们让数据一个接一个地放置，那么我们只要知道第一个数据所在地址单元就可以了（看图 2）如果第一个数据在 27H，那么第二、三个就在 28H、29H 了。所以利用堆栈这种方法来放数据可以简化操作

那么 51 中堆栈什么地方呢？单片机中能存放数据的区域有限，我们不能够专门分配一块地方做堆栈，所以就在内存（RAM）中开辟一块地方，用于堆栈，但是用内存的哪一块呢？还是不好定，因为 51 是一种通用的单片机，各人的实际需求各不相同，有人需要多一些堆栈，而有人则不需要那么多，所以怎么分配都不合适，怎样来解决这个问题?分不好干脆就不分了，把分的权利给用户（编程者），根据自己的需要去定吧，所以 51 单片机中堆栈的位置是可以变化的。而这种变化就体现在 SP 中值的变化，看图 2，SP 中的值等于 27H 不就相当于是一个指针指向 27H 单元吗？当然在真正的 51 机中，开始指针所指的位置并非就是数据存放的位置，而是数据存放的前一个位置，比如一开始指针是指向 27H 单元的，那么第一个数据的位置是 28H 单元，而不是 27H 单元，为什么会这样，我们在学堆栈命令时再说明。



其它的 SFR，我们在用到时再介绍。