致广大新老朋友

应广大朋友的要求，这次又对教程做了比较大的改动，把大家提出的错误的地方做了修改，另外加入了一些新的内容，因为现在自己在做产品，所以没太多的时间来解释大家学习中的问题，请大家自己到“电子爱好者网站—www.entui.com/bbs/index.asp”或我主持的网站“电子爱好者园地 www.fj136.com/bbs/index.asp”去提问，我会经常去那儿，看到了会给大家作解释。

发个广告，我们现在生产的产品，如果那位有需要，可记得一定要来找我哦！联系方法在下面，够多了吧，不要说找不到哦，呵呵！

室内全彩屏

＊中文简体条屏

＊中文繁体条屏

室内外条屏

前　言

基于本人学习单片机的痛苦经历，特编写本教程，以此献给广大的单片机初学者，希望您能从中受益。

作者提示：本教程乃最通俗易懂之单片机教材，如果您还是看不懂，请千万不要涉足此行，以免误入歧途，耽误您的前程*^*

拿到这本教程您首先就会想，什么是 IAP 教学法？是不是一种什么全新的教学方法？当然不是，我可没有那么大的本事，其实这只是我杜撰的一个名词，意思就是 In Applications Program（在应用中编程），当然这只是针对单片机教学，说法是否正确，还得您说了算。

至于为什么要提这种说法，那我倒想说几句。大家都知道，学习电子技术是一件非常无聊和枯燥的事情，为什么会有这种想法，就是因为我们传统的教学方法只重理论而忽略了实践，要一个人记住那些空洞而有无聊的理论知识实在不是一件容易的事，好在我们总算熬过来了，不管如何，也多多少少的学习了一些电子基础知识。

接下来我们应该进一步掌握些什么知识呢，凡涉足此行的朋友都知道，那就是单片机。不过这可不是一件容易的事，倒不是因为单片机很难学，而实在是我们身边很难找到一本专为单片机入门者而编写的教材。翻一下传统的单片机教材，都好象是为已经懂单片机的人而写的，一般总是以单片机的结构为主线，先讲硬件原理，然后是指令，接着讲软件编程，再是系统扩展和外围器件，最后举一些实例（随便说一点：很多书中的实例都是有问题？的），很少涉及单片机的基础知识，如果按照此种学习方法，想进行产品开发，就必须先把所有的知识全部掌握了才可以进行实际应用。孰不知，单片机不象模拟电路和数字电路那样，只要搞懂了电路原理，再按照产品要求设计好相应的电路就可以了。它是一种以简单的硬件结构，复杂而有灵活的软件系统来完成设计的通用性产品，不同的设计者只会使用其不同的功能，几乎没有人会把它的全部指令都使用起来，所以学习使用单片机只能靠循序渐进的积累，而不可能先把它全部掌握了再去做产品开发（当然天才就例外了*^*）。

基于以上原因，我们尝试了一种全新的单片机教学方法，打破传统的循序渐进式的教学方法，以单片机的应用为基础，结合基本的工业控制系统和实践工作中的具体应用，不分先后顺序，将各条指令贯串于一个又一个的实验中，通过所见即所得的实验来讲解各种指令的编程方法，顺便讲解相关的基本概念，使您尽快地熟悉单片机应用的基本步骤，掌握软件编程的基本方法。

本教程分为上、下两册，上册部分主要教您掌握单片机开发的基本步骤和软硬件的编程与设计，如果您学完了上册部分就能成为单片机的入门者，完全可以进行一般产品的开发；下册部分是单片机应用的提高部分，主要学习单片机的系统扩展（比如：ROM 和 RAM 存储器的扩展，并行口的扩展，串行口的扩展，A/D 和 D/A 与单片机的接口）以及相关开发工具和软件的使用（包括 KELL C51 的应用与调试技巧，硬件仿真器的使用）等等，如果您学完了下册部分，那就得恭喜您成为了单片机开发的高手了，不过单片机的技术是在不断的发展和提高的，您也不要太骄傲哦！

为了尽量把最新的单片机知识和应用成果收录进我们的教程，希望您能不吝赐教，共同来努力把我们的教程不断的改进和完善。还是那句题外话，技术是靠不断的积累和交流才会进步的，固封自守只会更加落后。

由于时间和精力的限制，我还是希望在您学习本教程之前，自己先熟悉一点相关的电子技术知识，特别是数字电路基础，这对您学习中碰到的相关概念会有很大的帮助。

目录

（上册）

（下册）

第一课 单片机的概述

因为我们的主要课程是单片机的应用，本来不想讲解单片机的历史与发展（这话说现状更确切些），但为了兼顾大多数朋友，我还是简单的介绍一下这方面的相关知识。

一．单片机的由来

单片机，专业名称—Micro Controller Unit(微控制器件)，它是由大名鼎鼎的 INTEL 公司发明的，最早的系列是 MCS-48，后来有了 MCS-51，我们经常说的 51 系列单片机就是 MCS-51（micro controller system），它是一种 8 位的单片机，8 位是什么意思，我们以后再讲。

后来 INTEL 公司把它的核心技术转让给了世界上很多的小公司（不过，再小也有几个亿的销售/年哦），所以世界上就有许多公司生产 51 系列兼容单片机，比如飞利浦的 87LPC 系列，华邦的 W78 系列，达拉斯的 DS87 系列，现代的 GSM97 系列等等，目前在我国比较流行的就是美国 ATMEL 公司的 89C51，它是一种带 Flash ROM 的单片机（至于什么是 Flash ROM，我在这儿先不作介绍，等以后大家学到相关的知识时自然就会明白），我们的讲座就是以该型号的单片机来作实验的。讲到这里，也许有的人会问：我平时在各种书上看到全是讲解 8031，8051 等型号的单片机，它们又有什么不同呢？其实它们同属于一个系列，只是 89C51 的单片机更新型一点(事实上,89C51 目前正在用 89S51 代替，我们的实验系统采用就是 89S52 的，兼容 89C52)。这里随便说一下，目前国内的单片机教材都是以 8051 为蓝本的，尽管其内核也是 51 系列的，但毕竟 8051 的单片机已经属于淘汰产品，在市场上也很少见到了，所以由此感叹，国内的高等教育是如此的跟不上时代的发展需要！这话可能会引起很多人的不满,所以大家别说是我讲的哦!!!

二．主要单片机的分类

接着上面的话题，再给大家介绍一下我们经常在各种刊物上看到的 AVR 系列和 PIC 系列单片机是怎么回事？以便让大家对单片机的发展有一个较全面的认识。在没有学习单片机之前，这是一个令很多初学者非常困惑的问题，这么多的单片机我该先学哪一种呢？

AVR 系列单片机也是 ATMEL 公司生产的一种 8 位单片机，它采用的是一种叫 RISC（精简指令集单片机）的结构，所以它的技术和 51 系列有所不同，开发设备也和 51 系列是不通用的，它的一条指令的运行速度可以达到纳秒级（即每秒 1000000000 次），是 8 位单片机中的高端产品。由于它的出色性能，目前应用范围越来越广，大有取代 51 系列的趋势，所以学完了 51 系列的，看来必须学会 AVR 的才行，可叹知识爆炸，人生苦短。说完了 AVR 的，再来说说另一种--PIC 系列单片机，它是美国 MICROCHIP 公司，唉，又是老美，叫微芯公司的生产的另一种 8 位单片机，它采用的也是 RISC 的指令集，它的指令系统和开发工具与 51 系列更是不同，但由于它的低价格和出色性能，目前国内使用的人越来越多，国内也有很多的公司在推广它，不过它的影响力远没有 51 系列的大，所以作为初学者，51 系列当然是首选。

以上几种只是比较多见的系列，其实世界上还有许多的公司生产各种各样的单片机，比如：MOTOROLA 的 MC68H 系列(老牌的单片机),TI 的 MSP430C 系列(极低功耗的单片机),德国的西门子 SIEMENS 等等，它们都有各自的结构体系，并不与 51 系列兼容。为了不搞大家的脑筋，这里就不介绍了，等大家入了门以后自己去研究它吧！我们还是回来了解一下 51 系列单片机到底是个什么东西，它有那些部分组成，请接着往下看：

三．单片机的结构及组成

单片机到底是一种什么 DD，它究竟能做什么呢？其实它就是一种能进行数学和逻辑运算，根据不同使用对象完成不同控制任务的面向控制而设计的集成电路，此话好象有点绕口，没关系，大家都应该知道我们经常使用的电脑吧，在电脑上，我们可以用不同的软件在相同的硬件上实现不同的工作。比如我们用 WORD 可以打字，用 PROTEL 可以设计图纸等等，单片机其实也是如此，同样的芯片可以根据我们不同的要求做出截然不同的产品，只不过电脑是面向应用的，而单片机是面向控制的，比如控制一个指

示灯的亮和灭，控制一台电机的启动和停止等等。

那么它的内部究竟由哪些部件组成的呢？大家都知道我们的电脑有很多的零件，比如 CPU（中央处理器），RAM（内存条），ROM（程序存储器，有点象硬盘），输入输出设备（并行口串行口）等等，在单片机中这些部件都有，而且还把它们全部做到了一块芯片上（这就是单片机名称的由来）。

讲到这里，您一定会想，这么多零件集成在一块芯片上，那为什么单片机的价格会这么便宜（89S51 每块才 10 元不到），其实原因很简单----功能有强弱，就象我们平时用的 PLC，控制一台数控机床要用 128 点的，而控制一台电机有几点的就足够了，另外这种芯片的产量很大，技术也非常的成熟，自然价格也就很低了。

说到这里，你是不是有点心痒了，也想学习学习，那么学习单片机究竟需要些什么设备，又要做哪些准备呢？对于一个初学者来说了解这些真的是很有必要哦，尽量化最少的钱毕竟是大家的想法嘛。

四．学习单片机的准备工作

首先您需要一台电脑，这是最基本的，配置嘛，P2 以上的就可以了；然后您需要一套开发单片机的软件，这个软件叫 KEIL C51，它是美国 Keil Software 公司专门为 MCS-51 系列单片机开发的第三方软件，它的免费测试版可在www.keil.com 上下载，也可以在各种单片机网站上下载，最新版本是 V6.23，安装时选择 Uvision2，虽然有 2K 代码的限制，但足以满足我们学习的需要；其次，您还需要一台编程器，它是一种把程序写进单片机芯片的设备，这种设备品种很多，操作也很简单，大家既可以买现成的产品（价格从 200 多元到 2000 多元的都有），也可以自己制作；有了这两样东西还不行，为了看到程序执行的结果，我们还需要一块实验板。

因为现在没时间，学习系统我也不做了，以前做的还有一些零件，大家如果有需要，我可以送给大家，具体的说明在电子爱好者园地上，有时间自己去看一下，网址www.fj136.com/bbs/index.asp

实验系统的介绍看下面的内容：

传统的单片机实验过程都是先用 KEIL C51 或其他的单片机开发软件把源代码汇编成 HEX 或 BIN 文件；然后用编程器把汇编文件烧写入单片机中；再把单片机插入实验板中，才能看到软件的执行结果。对于一个单片机初学者来说，不仅非常的麻烦，而且必须配置一套编程器和实验板，就目前市场上最便宜的编程器来说，投资也要 300 多元。如此一来，使得很多想学单片机，但又不想花太多钱的爱好者忘而却步。

好在现在出现了一种支持在线下载的单片机，只要满足一定的外部条件，就能够直接把汇编的程序下载到目标单片机中。经过实验，我们开发设计了这样的一套实验系统，它采用了一套集源代码编辑、软件汇编、程序下载于一体的专业软件，采用具有在线下载功能的 FLASH ROM 单片机 89S52，配合

本教程，可以完成教程中的每一个实验。这样既免去了您添置编程器和仿真器等设备的昂贵费用，又可以直接在实验板上看到程序执行的结果，更由于采用的是 FLASH ROM 的存储器，烧写次数可以达到 1000 次以上。产品具体电路图在教程的最后面附录中。

为了尽量降低大家的学习费用，我们采用了模块化的设计方法，在您学习上册时只需购买实验系统的主机部分，包括了 8 个发光二极管，2 个数码管，4 个功能按键，1 个蜂鸣器，一个串行芯片和成品外客，完全可以做上册中的每一个实验。

当您开始学习下册时可以再购买扩展模块，现在开发的扩展模块包括 A/D（TLC0831）和 D/A（MAX517）转换，外部 RAM（6264）和外部 ROM（29F020）存储器，16*16 汉字显示点阵，I2C 总线（24C01）和温度转换（DS18B20），日历时钟（DS1302）和液晶字符模块（T6963 驱动的 240*128 中文图形点阵）等。如果你把这些都学会了，那就得恭喜您，因为您已经基本学会了使用 MCS-51 系列单片机。

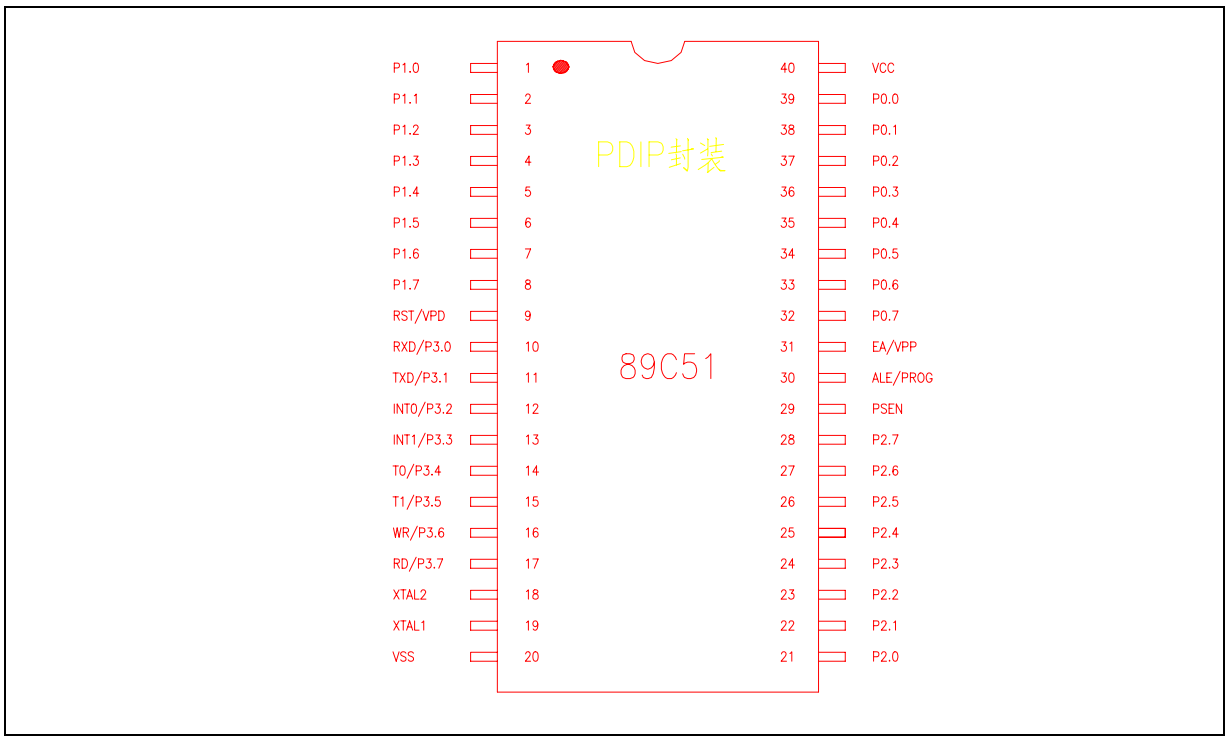
四. 第 1 课习题

1. 单片机的分类有几种？
2. 单片机与计算机有什么区别？

第二课 单片机硬件结构和开发过程

从这一课开始，我们就要正式开始学习 MCS-51 单片机了，前面我们曾经提到过单片机的内部结构是由 CPU、ROM、RAM 等等组成，它们的内部结构我们以后再讲，这一课让我们先来看看它的外部引脚（即硬件结构），看下面的图，这就是我们要实验用的 89C51 单片机的外部引脚图，给大家简单介绍一下。

一．单片机的引脚功能



1. VCC（40 脚）：接电源+5V。
2. VSS（20 脚）：接地，也就是 GND。
3. XTAL1（19 脚）和 XATL2（18 脚）：接振荡电路。

单片机是一种时序电路，必须有脉冲信号才能工作，在它的内部有一个时钟产生电路，有两种振荡方式，一种是内部振荡方式，只要接上两个电容和一个晶振即可；另一种是外部振荡方式，采用外部振荡方式时，需在 XTL2 上加外部时钟信号（详细的内容将在以后的课程中专门介绍）。

4. PSEN（29 脚）：片外 ROM 选通信号，低电平有效。
5. ALE/PROG（30 脚）：地址锁存信号输出端/EPROM 编程脉冲输入端。

至于它们的作用我们暂时不去管它，等以后学到相关的知识时再来研究它。这也许就是本教程区别于其他教材的最大特点----先实践后理论，尽量用实验结果来总结理论知识，因为单片机是一种通用的产品，它的功能设计是为了满足大多数使用者的要求，换句话说，不同的使用者只会使用其相关的功能，几乎不可能把全部的功能都用起来，因此我们完全不必象学习其他电子技术那样，把单片机的全部知识都搞懂了再去开发产品。这话前面好象说过了，打住!!!

6. RST/VPD（9 脚）：复位信号输入端/备用电源输入端。

什么是复位信号，为什么要加复位信号？当然也暂时不去管它。

7. EA/VPP（31）：内/外部 ROM 选择端。

在 30 脚、9 脚的功能上不知大家注意没有，都有一个/，什么意思呢？这是引脚的第二功能，也就是说，该引脚既可以作前面的功能，也可以作后面的功能，至于它是如何工作的，我们暂时也别去研究它。

8. P0 口（39-32 脚）：双向 I/O 口。
9. P1 口（1-8 脚）：准双向通用 I/O 口。
10. P2 口（21-28 脚）：准双向 I/O 口。
11. P3 口（10-17 脚）：多用途口。

I/O 就是英文 IN/OUT 的缩写，这些引脚的功能想必大家也都明白了，（就是输入/输出的意思），这 32 个 I/O 口就是留给我们作连接外围电路用的，那么它们之间有些什么不同呢？这个问题稍微有点复杂，我们将在稍后再来专门学习。现在我们先来往下看：

二．单片机的电路连接和开发过程

看最后面的附图，这就是我们做实验用的电路图，想必大家都能看得懂吧。接下来就让我们通过一个实验来看看单片机是如何工作的？我们的实验是让一个 LED 灯亮起来，亮哪一个？这就随便你了，比如我们就让 LED1 亮起来吧，仔细看一下电路图，LED1 接在什么地方呢？接在单片机的 P1.0 的引脚（也就是 1 脚）上，那么按照该电路图的连接方法，当 1 脚为高电平时，LED1 是不亮的；只有当 1 脚为低电平时，LED1 才会亮起来，怎样才能让 1 脚由高电平变为低电平呢？我们让人做事，就必须对她说一声，也就是发布命令，想让单片机工作，也得发布命令，不过在计算机中那叫指令，我们要让 1 脚变为低电平的指令是 CLR P1.0（让 1 脚变为高电平的指令是 SETB P1.0），这就是我们通常所说的源代码，（这是我们实验的第一步—源代码编辑）；怎么做呢？我们首先得打开实验软件，屏幕出现一个浏览器的软件窗口，点击左边的扩展实验，选中实验 16—自动温度控制器，再点击工具栏里的调试按钮，弹出一个记事本对话框，写入 CLR P1.0;(注意☺分号必须在英文状态下输入)，输入完毕后选择文件→保存即可；那么单片机能读懂这条指令吗？当然不能，接下来我们还有一件事情要做，就是把这句指令翻译成单片机能读懂的东西，单片机能读懂什么呢？它其实只懂一样--就是数字，因此，我们就把 CLR P1.0 翻译成 C2H,90H，至于为什么要翻译成这样，这当然是 INTEL 公司规定好的，我们就不需要去研究它了。这个过程我们叫作编译，（这是我们实验的第二步），那么指令是怎么编译过来的呢？这就得靠专业的软件了，我们做实验使用的软件就有此功能，只要点击工具栏上的编译按钮，稍等片刻即出现一个编译信息窗口，如果编译通过就会有编译完成，结果如下：0 个警告，0 个错误的编译信息，如果编译错误则会出现编译错误的信息，并提示错误的行号；编译完了之后通常要进行程序仿真（这是第三步），当然我们的实验程序很简单是不需要仿真的；接下来怎么才能把编译通过的指令写入单片机中呢？这通常需要借助于一种硬件工具，叫编程器（也叫烧录器），不过我们的实验板采用的是具有串行下载功能的单片机，所以您只要直接点击快捷工具栏上的下载按钮，程序就进入了实验板（这是第四步—编程）。自此就完成了单片机实验的全过程。

全部工作结束后，我们看到了什么？接 P1.0（1 脚）的 LED1 亮了起来；改变源代码，变成 SETB P1.0; 进行编译，下载，看看结果是不是 LED1 不亮了。怎么样，不难吧!!!

最后让我们来思考一个问题，当我们用编程器把编译后的指令写入单片机时，单片机就开始执行这条指令，那么这条指令就一定在单片机内部的某个地方，它究竟在哪里呢？单片机的内部结构又是怎么样的呢？这将是我们第三课要讨论的内容—单片机的内部结构（一）……半导体存储器。

三．本课总结

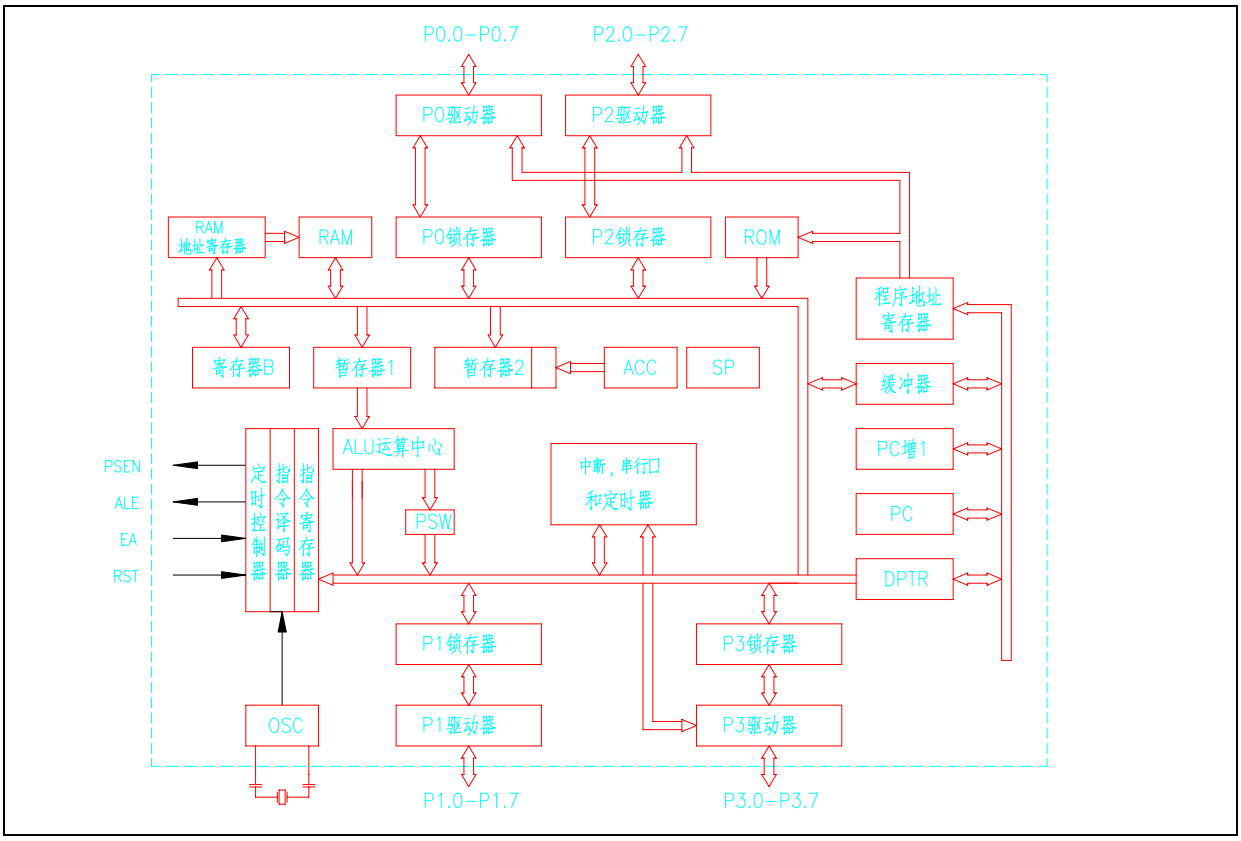
本课主要讲述了单片机实验的整个过程，这个过程包括第一步—编辑源代码，第二步—编译源代码，第三步—程序仿真，第四步—芯片烧写（亦称编程），希望大家记住这四步，这是非常关键的哦；单片机的硬件原理其实并不复杂，本来嘛，单片机的设计就是为了通用和灵活，所以开发单片机最重要的就是软件的编写，以后我会慢慢地教大家学习更多的软件知识。

四．第 2 课习题

1. 89C51 的电源电压是多少伏？
2. 熟练掌握编译软件的使用方法。

第三课 单片机的内部结构（一）

单片机的内部究竟有哪些部分组成的，它们都有些什么作用呢？让我们看下面的图：



这就是单片机的内部方框图，我们先来了解其中的 ROM 存储器吧：

一．半导体存储器 ROM

1．几个基本概念

上一课我们讲到了把编译后的指令下载到单片机后这条指令一定在单片机内的某个地方，那么它究竟在哪里呢？原来它就放在一个叫程序存储器的地方，英文名称 ROM（全称为 Read Only Memory），叫只读存储器。它是一个什么东西呢？在讨论这个问题之前，让我们先来看几个物理现象：

（1）数和物理现象的关系

不知大家是否还记得，在学习数字电路时我们曾用一盏灯的亮和灭来表示电平的高和低，即用“1”来表示高电平，用“0”来表示低电平，如果现在有两盏灯那它会有几种状态呢？请看下面的表：

¤	¤
0	0
0	1
1	0
1	1

两盏灯的组合就是四种状态：00，01，10，11。这样看来灯的亮和灭这种物理现象同数字确实有着某种联系，如果我们把它们按一定的规律排好，那么电平的高或低就可以用数字来表示了，换句话说：不同的数字可以代表不同数量灯的电平高或低。比如：0000，0001，0010，0011，0100，0101，0110，0111，1000，1001，1010，1011，1100，1101，1110，1111 这十六种组合就可以代表四盏灯的状态，能理解吗？

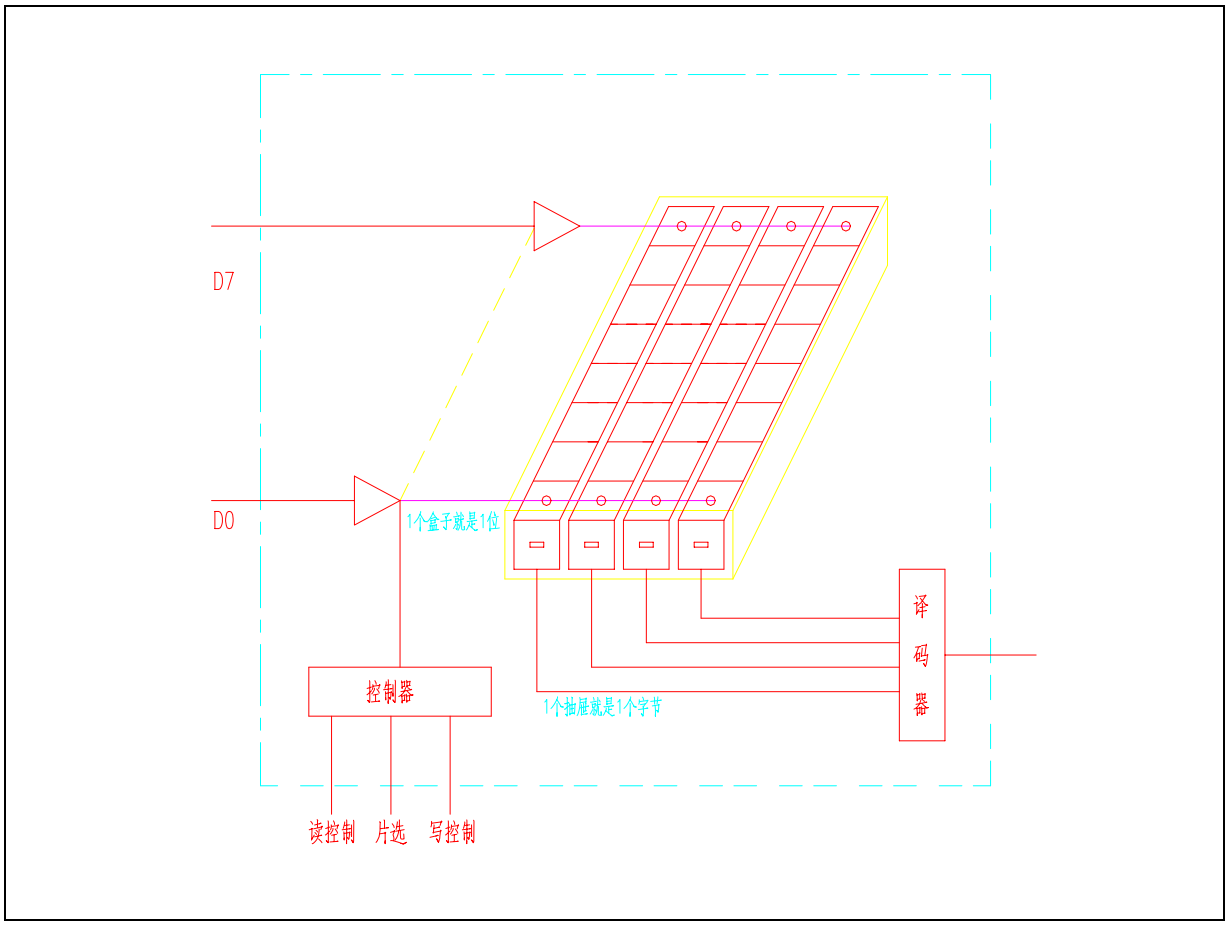
（2）位及字节的含义

在单片机中，一盏灯（实际上是一根线）我们称它为一位，它有两种状态（“0”或“1”），分别对应电平的高或低，它是单片机最基本的数量单位，用 BIT 来表示。8 盏灯（八根线）有 256 种状态，这 8 盏灯（也就是 8 位）我们把它称为一个字节，用 BYTE 表示。至于为什么要怎么规定，这就不需要你我操心了，我们只要记住就可以了。那么单片机是如何来储存这些数字所代表的字节的状态的呢？接着往下看：

2．半导体存储器的工作原理

（1）存储器的内部构造

看下面的图，这就是半导体存储器的结构简图：(图中有 4 个字节)



（2）存储器的工作原理

存储器就是用来存放数据的地方，它其实是利用电平的高或低来存放数据的，也就是说，它实际上存放的是电平的高或低的状态，而不是我们所习惯上认为的“1234”这样的数字。那它是如何工作的呢？看上面的图，这就是存储器的内部结构示意图，一个存储器就象一个小抽屉，一个小抽屉里有 8 个（也就是单片机的 8 位）小盒子，每个小盒子用来存放 1 位“电荷”，电荷通过与它相连的电线传进来或释放掉，至于电荷在小盒子里是怎样存放的，这就不用我们操心了，您可以把电线想象成水管，小盒子里的电荷就象是水，那就好理解了，存储器中的 1 个小抽屉我们把它称之为 1 个“单元”，相当于 1 个字节，而 1 个小盒子就相当于 1 位。

有了这么一个构造，我们就可以开始存放数据了，比如我们要放进一个数据“00011010”，我们只要把第 2 号、第 4 号和第 5 号小盒子里存满电荷，而其它小盒子里的电荷给放掉就行了。可是问题又出来了，一个存储器有好多相同的单元，线是并联着的（看 D7-D0），在放入电荷的时候，会将电荷放入所有的字节单元中，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字节单元，都只能放同一个数，这当然不是我们所希望的。因此，我们要在结构上稍作变化，看上面的图，在每个单元上有根线与译码器相连，我想要把数据放进哪个单元，就通过译码器给哪个单元发一

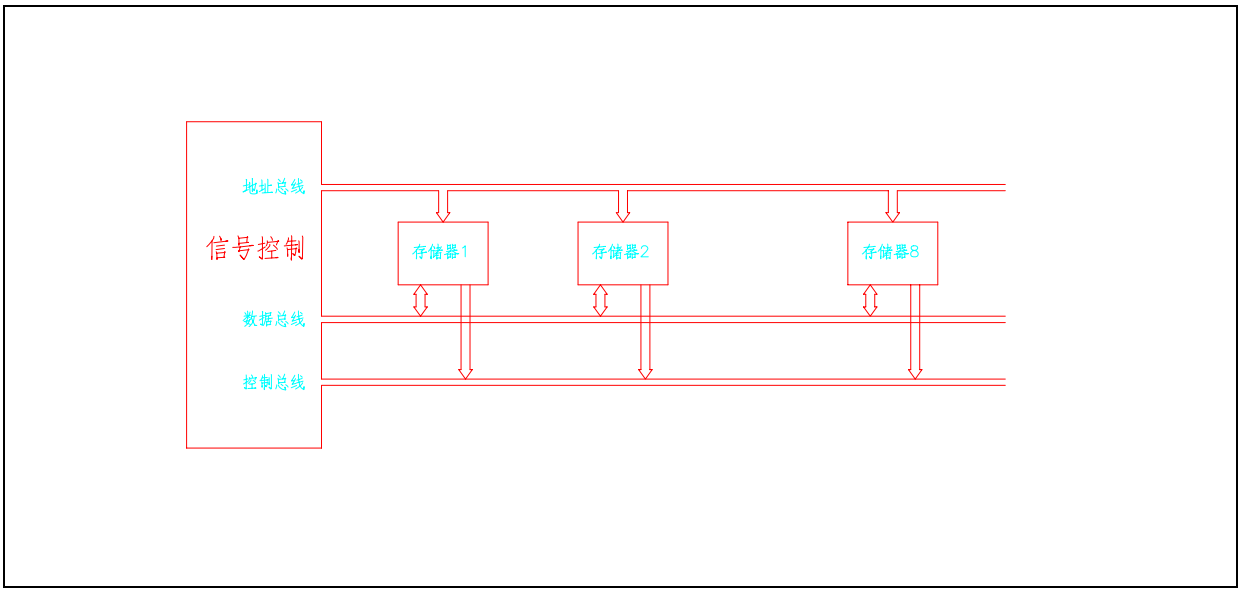
个信号，由译码器通过这根线把相应的开关打开，这样电荷就可以自由地进出了。那么这样是不是就能随意地向存储器写入或者读出数据了呢？其实还不能，继续看上面的图，与 D7-D0 相连的还有一个控制器，它是用来干什么的呢？这根线叫写入/读出控制线，当我们向存储器写入数据时，必须先把这个开关切换到写入端；而要读出数据时，就得先把开关切换到读出端；而片选端则是为了区分不同的存储器设置的。这里没搞明白，没关系，后面还有介绍，先让我们来看看译码器是如何工作的？

3．半导体存储器的译码

简单介绍一下：我们知道，1 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 8 种；256 种状态又需要几根线代表？自己想一下，是不是 8 根线。

4．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：送入每个字节的 8 根线又是从什么地



方来的呢？它就是从单片机的外部引脚上接过来的，一般这 8 根线除了接一个存储器之外，还要接其它的器件，这样问题又来了，这 8 根线既然不是存储器和单片机之间专用的，如果总是将某个单元接在这 8 根线上，就不行了，比如这个存储器单元中的数值是“11111111”，另一个存储器的单元是“00000000”，那么这根线到底是处于高电平，还是低电平？所以我们必须让它们分离。办法当然也简单，当外面的线接到集成电路的引脚上来后，不直接接到各单元去，而是在中间加一组开关。这组开关就是前面提到的控制器（看前面的图），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，再让开关切换到相应的位置就行了。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，先由控制器选中该片，然后发出相应的写信号，开关切换到相应的位置，并将传过来的数据（电荷）写入片中；如果要读信号，先选中该片，然后发出读信号，开关也切换到相应的位置上，数据就被送出去了；另外读和写信号还同时受到译码器的控制，由于片选端的不同，所以虽有读或写信号，但没有片选信号，所以另一个存储器就不会“误会”而开门，造成冲突，那么会不会同时选中两个存储器呢？只要是设计好的系统就不会，如果真的出现同时选中两个存储器的话，那就是电路出故障了。

如此看来，存储器要想写入或者读出数据还真是不容易，不过好在这些都是由计算机自动完成的，不需要我们去操心。从上面的介绍中我们已经看到，用来传递数据的 8 根线并不是专用的，而是很多器件大家共用的，所以我们把它们称之为数据总线（总线英文名为 BUS），即公交车道，谁都可以走；而 16 根地址线（51 单片机共有 16 根地址线，这些以后会讲解，这里不必死记硬背）也是连在一起的，我们把它们称之为地址总线，看上面的图。

对于本小节的内容，如果您一时还无法理解，没有关系，继续往下看好了，我们会在以后的课程中再来详细的介绍，这里你只要稍微的了解一下就可以了！

5. 半导体存储器的分类

第一课中我们提到过，89C51 是一种带 Flash ROM 的单片机，什么是 Flash ROM？它到底是一种什么东西呢？ROM 我们已经知道，是只读存储器，所谓只读，从字面上理解那就是只可以从里面读出数据，而不能写进去，它类似于我们的书本，发到我们手里之后，我们只能读里面的内容，不可以随意更改书本上的内容。ROM 就是单片机中用来存放程序的地方，前面我们下载到单片机的指令就放在这个地方。讲到这里大家也许会感到困惑，既然 ROM 是只读存储器，那么指令又是如何进入其中的呢？其实所谓的只读只是针对工作情况下而言，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，只要让存储器满足一定的条件就能把数据预先写进去，这个道理也很好理解，书本拿到我们手里是不能改了，但当它还是原材料--白纸的时候，我们完全可以由印刷厂把内容印上去嘛。前面的编程就是这么回事！

Flash ROM 是一种快速存储式只读存储器，这种程序存储器的特点是既可以电擦写，而且掉电后程序还能保存，编程寿命可以达到一千次左右，所以我们的实验系统是可以反复烧写的，你尽管使用。目前新型的单片机都采用这种程序存储器；当然，除了这种程序存储器外，还有两种早期的程序存储器产品，简单介绍一下：PROM，EPROM 和 EEPROM，PROM 称之为可编程只读存储器，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，所以它只能写一次，要是写错了，就报废了，习惯上我们把带这种程序存储器的单片机称为 OTP 型单片机，如果您的产品批量生产，又要求价格比较低的话，带这种程序存储器的单片机是非常合适的；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，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—几十次吧，电脑上的 BIOS 芯片采用的就是这种结构的存储器；EEPROM，前一种存储器的擦写要用紫外线，而这种存储器可以直接用电擦写，比较方便数据的改写，它有点类似于 FLASH 存储器，但比 FLASH 存储器速度要慢，现在新型的外部扩展存储器都是这种结构的。

有关这几种程序存储器的使用和原理，我们将在下册中详细的介绍，这里就不多讲了。总之一句，不管哪种程序存储器，它们的作用都只有一个----就是用来存放程序（也就是我们为单片机编写的指令）。

了解了 ROM，让我们再来简单讲讲另一种存储器，叫随机存取存储器，也叫内存，英文缩写为 RAM（Random Access Memory），它是一种既可以随时改写，也可以随时读出里面数据的存储器，类似于我们上课用的黑板，可以随时写东西上去，也可以用黑板擦随时擦掉重写，它也是单片机中重要的组成部分，单片机中有很多的功能寄存器都与它有关，详细内容后面再讲。

二．本课总结

本课主要讲述了单片机的两种半导体存储器—只读存储器 ROM 和随机存储器 RAM 的工作原理，它们是单片机的重要组成部分，了解它的内部结构对我们学习单片机是很有帮助的。不过如果您一时对本课的内容还无法搞得很明白，也没有关系，随着学习的深入，我们还会慢慢地讲解相应的基础知识，可千万不要放弃哦？我在没有学会单片机之前也是如此囫囵吞枣的。

三．第 3 课习题

1. 半导体存储器分为几大类？
2. ROM 存储器的作用是什么？
3. 什么是位？什么是字节？
4. 为什么 8 根线在单片机中会有 256 种状态？它是如何出来的？
5. 89C51 的 ROM 有多少字节的容量？

第四课 单片机的内部结构（二）

上一节课我们讲了半导体存储器 ROM 和 RAM 的内部结构，大家是不是觉得有些枯燥了，这一课让我们先来做一个实验：

一．LED 灯闪烁的实验程序

还记得第二课中的实验吗？这个实验在实际应用中太没有意义了，接下来我们要让 LED1 不断的闪烁，就象高楼上或者大海中用的航标灯，这个实验可是非常经典的，几乎所有的单片机实验都要提到，那么怎样才能让 LED1 不断的闪烁呢？实际上就是让它亮几秒，再灭几秒，也就是让 P1.0 交替地输出高电平和低电平，怎样来实现这个功能，按照前面所学的知识，我们写出下面的程序：CLR P1.0；SETB P1.0；编译后下载到单片机……

结果不行，为什么？这里有两个问题：首先计算机执行指令的速度很快，执行完第 1 条指令后 LED1 是灭了，但在极短的时间内又去执行了第 2 条指令，LED1 又亮了，我们根本无法看到灯曾经灭过；第二个问题是当执行完第 2 条指令后，不会再去执行第 1 条指令了，因为单片机执行指令的过程是一条一条地顺序执行的。

如何解决这两个问题呢？我们可以作如下的设想：第一，执行完第 1 条指令后让单片机延时一段时间（几秒或零点几秒），然后再去执行第 2 条指令，这样就可以看到 LED1 曾经灭过了；第二，让单片机执行完全部指令后再返回去执行第 1 条指令，如此不断的循环就可以达到我们的要求了。

实验程序如下：

主程序

```
MAIN:SETB  P1.0        ;（1）
       LCALL  DELAY    ;（2）
       CLR  P1.0       ;（3）
       LCALL  DELAY    ;（4）
       LJMP  MAIN      ;（5）
```

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）
```

发现许多朋友很聪明，喜欢把这里的内容复制了直接粘贴到实验系统中，这对你的学习很不利，所以现在的 PDF 文档我把它加密了，看你再偷懒！呵呵，别怪我，我也是为了你好*^*

还记得软件的使用方法吗？调试，写入源代码，编译，下载到单片机，看看是不是我们想要的结果……

在分析这段程序之前，先来说明几个标点符号的意义：1.分号在这里起一个分隔符的作用，表示这条指令到此为止；2.括号内的数字在这里是为了解释程序用的，实际的编译过程中是没有意义的，也就是说没有也是一样的，只是为了程序的可读性更强，我们一般会在分号的后面加上程序的注释文字（后面我们会用到）；3.特别☺：程序中的标点符号只能在英文状态下输入，当使用中文输入时，必须切换到半角状态，不然编译软件会出错。

接下来我们分析一下这段程序：按照我们的要求，第 1 条，让灯灭，第 2 条应该是延时，第 3 条是让灯亮，第 4 条和第 2 条一样也应该是延时，第 5 条应当返回去执行第 1 条指令。看一下上面的程序，第 1 条我们已经懂了，是让 LED1 灭，第 2 条和第 4 条我们等一下讨论，第 5 条是 LJMP MAIN，LJMP 是

一条指令，意思是转移，转移到什么地方去呢？看一下 LJMP 后面跟着什么，是 MAIN，什么地方有 MAIN，在第 1 条指令的开头就是 MAIN，所以第 5 条指令的意思就是跳转到 MAIN（即第 1 条指令处继续执行），如此一来，就不断地重复执行这些指令。那么 MAIN 又是什么意思呢？它实际上是我们为这段程序起的一个名称，专业术语叫标号，既然是一个名称那可不可以用 JIGUO，CHINA 等等的其他名字呢？当然可以，这完全取决于您的需要（☺：不过也有一些是不能采用的，我们以后会讲）。再来分析第 2 条和第 4 条指令，看看它们是如何实现延时的？LCALL　DELAY，LCALL 也是一条指令，这条指令叫做调用子程序指令，看看 LCALL 后面跟着的是什么--DELAY，哪里有 DELAY，在第 6 条指令的开头，很显然这也是一个标号，这条指令的作用就是当执行到这条指令时就转去执行 LCALL 后面标号所在处的程序，如果在执行程序时遇到 RET 指令（RET 叫返回指令），就返回到 LCALL 指令的下面一条（即第 3 条指令）处继续执行，在第 9 条指令后确实有 RET 指令，那么在执行完第 2 条指令后就应该去执行第 6.7.8.9 条指令，之后遇到第 10 条指令：RET，执行完这条指令后就回去执行第 3 条指令，将 P1.0 清零，也就是让 LED1 亮，然后再去执行第 4 条指令，执行完后又回到 6.7.8.9.10 条指令，最后执行第 5 条指令：LJMP　MAIN，也就是我们刚才说的跳转到第 1 条，将 P1.0 置位，就是让 LED1 灭掉。如此周而复始，LED1 就不断的闪烁。好好理解这段文字，务必把它搞清楚!!!

从标号 DELAY 处（即第 6 条）开始到 RET 的这一段指令我们称之为子程序，它是一段延时程序，至于延时多长时间，我们会在下一课中学习。程序的最后一条是 END，它不是指令，它只是告诉编译软件整个程序到此结束了，它叫“伪指令”。在大家以后的编程中，写完程序都要加上这一条。

在上面的程序中我们知道了从标号 DELAY 开始的子程序是一段延时程序，那么它又是如何工作的呢？在了解它的工作过程之前我们必须先知道其中的一些符号，就从 R7 开始吧，它是单片机内部的一个重要组成部分，叫工作寄存器，什么是工作寄存器？下面我们就来讲解这个问题：

二．工作寄存器

上一课我们已经讲过，在单片机中有许多的功能寄存器和半导体存储器 RAM 有关，那么工作寄存器又属于哪一部分呢？它是用来干什么的呢？要搞清楚这个问题，让我们先从日常生活中的一个例子说起，比如我们要做一道数学题 123+456，您会马上得出答案：579，接下来再看一道题：123+456+789，要你马上得出答案就不那么容易了，通常我们会怎么做呢？一般总是先把 123+456 的结果 579 写在一张纸上，然后再算 579+789=1368，这 1368 就是我们想要的最终结果，而 579 只是为了得到最终结果而暂时记下来的中间结果，单片机中做运算和我们生活中做运算一样，也需要把中间结果放在某个地方，那么计算机把它放在哪儿呢？前面我们提到的 ROM（只读存储器）中，不行！因为 ROM 是用来放程序的，它只能写进去，不能读出来（再次提醒一下，这只是相对而已），所以只能放在单片机的另一个区域—RAM 中（即随机存取存储器）中。R7 就是 RAM 区域中划出的一部分。知道了 R7，接下来让我们来分析一下这段子程序（延时程序）。

三．LED 灯闪烁程序子程序的分析

首先看第 6 条，MOV R7，#250，这也是一条指令，意思是传递数据。我们知道在日常生活中，要传递一件东西就必须要有一个传递者，一个接受者和被传递的东西，那么在单片机中是怎么区分它们的呢？在这条指令中，R7 是接受者，250 就是要传递的东西（单片机中要传递的东西当然是数字了），这里传递者被省略了（顺便提一下，并不是每条指令都能省略的，事实上大部分的指令都要有传递者），这样一来，这条指令的意思也很清楚了：就是把 250 这个数传递给 R7 这个工作寄存器（也就是把 250 这个数送入 R7 中），这样执行完这条指令后 R7 中的值就应该是 250，我们可以用 DBG8051 这个软件来验证一下，看是不是符合（这个软件的使用很简单，大家可以预先学一下）。比如，我们写下面的指令：

```
MOV　R7，#01；
MOV　R6，#02；
```

输入后按 F8，看看右边的特殊/工作寄存器窗口中 R6，R7 的值是不是 01H，02H（注意：实际显示的值是十六进制数，由于我们输入的十进制数，为了直观的看到执行结果，所以数值不要太大了。

这里还有一个问题，不知大家注意没有，在 250 这个数的前面有个#，它是什么意思呢？这个#就说明 250 是一个被传递的数的本身，而不是传递者（这里面是有区别的，我们以后会讲到）。看懂了 MOV

R7，#250，那么 MOV　R6，#250 也应该很清楚了。

接着看第 8 条 DJNZ　R6，D2，这又是另一条指令，我们来看一下 DJNZ 后面跟着什么，一个是 R6，一个是 D2，R6 我们已经知道了，再找一下 D2，D2 在本行的开头，我们已经学过，它是标号。那么这条指令是怎么执行的呢？它的执行过程是这样的：它将后面的值（即工作寄存器 R6 中的值）减 1，然后查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，转移到什么地方去呢？大家应该明白了，实际上这条指令的执行结果就是在原地转 250 次；当 R6 中的值等于“0”之后，程序就去执行第 9 条指令，也就是 DJNZ　R7，D1，大家自行分析一下这条指令的结果（是不是转去执行 MOV　R6，#250，同时 R7 中的值减 1），这段子程序的最终执行结果就是 DJNZ　R6，#250 这条指令被执行了 250*250=62500 次，执行这么多次干吗？就是为了延时。

四．本课总结

大家可以改变一下 MOV　R6，#250 这条指令中的值（注意☺：不能大于 255，为什么，以后会讲到）或者改变一下标号的名称，看是不是符合上面的分析。接下来提一个问题：通过实验我们看到了 LED1 在闪烁，是因为 DJNZ　R6，#250 这条指令被执行了 250*250=62500 次，执行那么多次究竟需要多长时间呢？下一课我们再来专门讨论这个问题。

这里有必要介绍一下 DBG8051 这个软件，它是一个专为 8051 单片机设计的仿真软件，配合 MON51 仿真机能进行 51 单片机的仿真，拥有这样一套设备在过去可是非常奢侈的，不过现在已经很少有人使用它了，原因是目前市场出现了许多兼容 KEIL C51 的仿真器，它们的功能更先进，MON51 只能属于淘汰产品。不过作为单片机初学者，使用 DBG8051 还是很有意义的，相比其他的开发工具，它的使用比较简单，我们可以用它来理解单片机的内部结构和程序的执行结果，在我们实验套件的随机光盘中，有这个软件，希望大家有时间好好的看一看。

五．第 4 课习题

1. 什么是主程序？什么是子程序？
2. 标号的含义是什么？
3. 单片机是如何执行程序的？
4. 工作寄存器属于 ROM 单元还是 RAM 单元？
5. 在实验中如果没有 RET 指令会出现什么情况？
6. 理解指令 LCALL、LJMP、DJNZ 的意义。
7. 掌握 DBG8051 软件的使用方法。

第五课 单片机的内部结构（三）

上一课中，我们提到了 DJNZ R6，#250 这条指令被执行了 250*250=62500 次，就产生了延时，那么这个时间是多少呢？它又是如何计算出来的呢？这一课就来讨论这个问题。

一． 单片机的时序

1．时序的由来

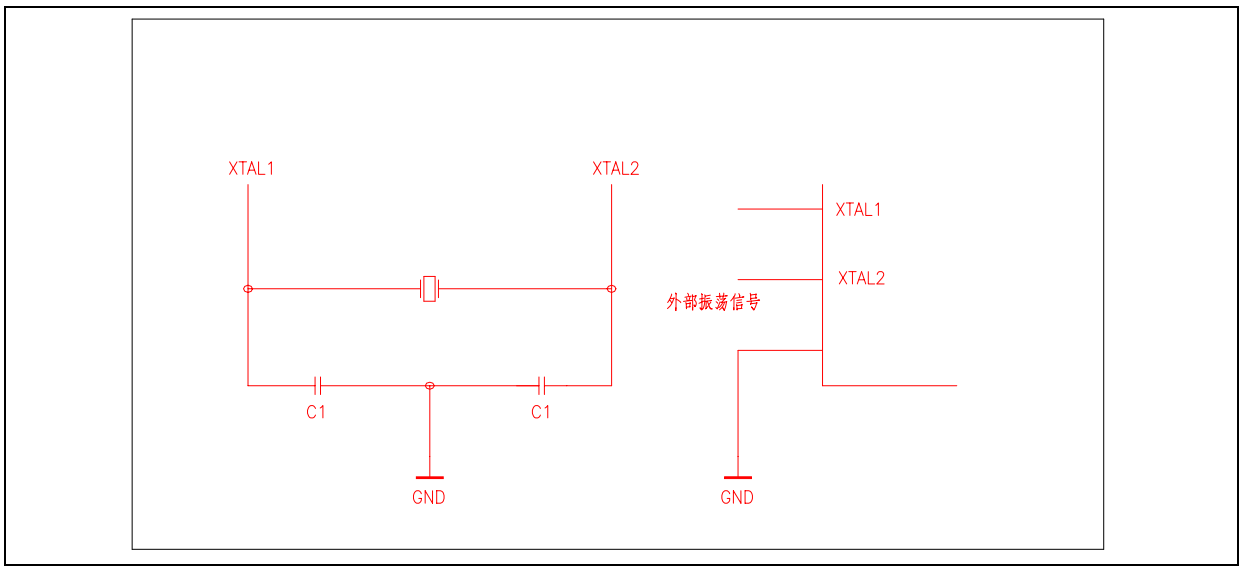
我们已经知道单片机执行指令的过程就是顺序地从 ROM（程序存储器）中取出指令一条一条的顺序执行，然后进行一系列的微操作控制，来完成各种指定的动作。它在协调内部的各种动作时必须要有一定的顺序，换句话说，就是这一系列微操作控制信号在时间上要有一个严格的先后次序，这种次序就是单片机的时序。这就好比我们学校上课时用的电铃，为了保证课堂秩序，学校就必须在铃声的统一协调下安排各个课程和活动。那么单片机的时序是如何规定的呢？接着往下看：

2．时序的周期

计算机每访问一次存储器的时间，我们把它称为一个机器周期，它是一个时间基准，就象我们日常生活中使用的秒一样，计算机中一个机器周期包括 12 个振荡周期，什么是振荡周期？一个振荡周期是多少时间？振荡周期就是振荡源的周期，也就是我们使用的晶振的时间周期，一个 12M 的晶振，它的时间周期是多少呢，电子技术过的朋友应该不难算出（T=1/f），也就是 1/12（微秒），那么使用 12M 晶振的单片机，它的一个机器周期就应该等于 12*1/12（微秒），也就是 1μS。

在 89C51 单片机中，有些指令只要一个机器周期，而有些指令则需要两个或三个机器周期，另外还有两条指令需要 4 个机器周期，这也不难理解，你在家擦地板的话总比擦桌子的时间要长，不过我可是大男子主义，从来不做家务的。开句玩笑!!! 如何衡量指令执行时间的长短？我们就要用到一个新的概念：指令周期—即执行一条指令所需的机器周期，INTEL 公司规定了每一条指令执行的机器周期，当然这不需要我们非把它记住，不过在这里 DJNZ 指令我们是要记住的，它是双周期指令，执行一次需要两个机器周期，即 2μS。（12M 晶振的话），回到我们上一课的实验，延时的时间就应该算出来了吧，是 62500*2μS=125000μS，也就是 125mS。这么大的数字也就 0.125S，怪不得 LED1 闪烁的这么快。（这里给大家出个题目：在上一课的实验中，如何延长闪烁的时间？想想看，怎么做？当然，不会也没关系）。

二．单片机的时钟电路



大家已经知道，单片机是在一定的时序控制下工作的，那么时序和时钟又有什么关系呢？时钟是时序的基础，单片机本身就如同一个复杂的同步时序电路，为了保证同步工作方式的实现，电路就要在唯一的时钟信号控制下按时序进行工作。那么单片机内的时钟是如何产生的呢？

1． 内部时钟电路

在 MCS-51 单片机的内部有一个高增益的反相放大器，其输入端为引脚 XTAL1（19 脚），输出端为 XTAL2（18 脚），我们只要在外部接上两个电容和一个晶振，就能构成一个稳定的自激振荡器，它的内部电路的工作原理就不介绍了，这里主要讲一下电容和晶振的选择，看上面的图，晶振的大小与单片机的振荡频率有关，我们到串行接口时再详细讲解，电容的大小影响着振荡器振荡的稳定性和起振的快速性，通常选择 10-30P 的瓷片电容或校正电容；另外在设计电路时，晶振和电容应尽可能的靠近芯片，以减少 PCB 板的分布电容保证振荡器工作的稳定性，提高系统的抗干扰能力。

2. 外部时钟电路

除了内部时钟方式外，单片机还可以采用引入外部时钟的振荡方式，什么时候需要采用外部时钟方式呢？当我们的系统由多片单片机组成时，为了保证各单片机之间时钟信号的同步，就应当引入唯一的公用的外部脉冲信号作为各单片机的振荡脉冲，此时应将 XTAL2 悬空不用，外部脉冲信号由 XTAL1 引入，如上右图所示。这是大虾们的作品，在此就不介绍了。

三．本课总结

本课的内容比较少，我就罗嗦一下，讲几句题外话。我是一个只有初中毕业，没有读过多少书的人，从小就爱好无线电，记得很小的时候，当通讯兵的父亲带回来几本电子方面的书籍，从此就迷上了无线电，那种痴迷程度决不亚于现在的小孩迷恋游戏机，至今仍然清楚的记得，曾经因为装成功一台 6 管收音机而兴奋的几天几夜没睡好觉。那个时候，我国的电子工业还刚刚起步，买一个 3AX31 的三极管都要特地跑到市区，而且价格奇贵，几乎要用去一个月的零化钱，当时最愿意去的地方就是上海的虬江路电子旧货市场，因为在那里可以淘到好多旧的电子元件。初中毕业以后，在当地根本就找不到一家电子企业，只好在镇上开了一家电器修理店，也就是这几年，边干边做地学了不少在今天已根本无法再学得进去的“电子啊，空穴啊，移位啊，寄存啊”等等理论知识，由于身边没有一个可以请教的老师，为了加深学习的印象，所以只好一边做实验一边学理论，尽管进度很慢，但效果竟然还不错，好在当时搞家电修理的收入还可以，加上没有家庭负担，也就这么过来了。

随后的几年，做过工人，也当过老师，但更多的时间是在搞技术开发，这些年来，看到很多的昔日同学靠导腾房地产或者做生意发了财，可自己依然还在这个领域默默无闻的钻研着，但我还是没有后悔，也从来没有想过改行，因为电子技术那众多迷人而未知的领域常常会使我深深地陷入其中，以至无法自拔，也感叹自己搞了这么多年，还只是一个入门者。

现在老是听到有些年轻的朋友说我要速成单片机，速成 C 语言，速成什么什么的，每当我看到或听到这些话的时候，总有一种说不出的滋味。现在的社会，什么都讲究个效率，这本来没有错，但学一项技术也能速成，实在让人有点不知道说什么好。就单片机而言，即使你现在只有 15，6 岁，也很有天赋，想把现在的几种主流单片机都搞懂并很好的应用到实践中去，没有个几年恐怕也难，更何况单片机的技术是在不断发展的，你想跟也来不及。

不过，话又说回来，我不是要打击大家的学习积极性，单片机是一种非常宽泛的技术，它的设计是为了满足大多数的需要，换言之，即使你并没有把全部的知识都理解得很深透，或者说没有把每种单片机都搞懂，也没关系，你一样可以在实际的产品开发中应用它，因为几乎没有一个产品会把全部的指令都用起来。

好了，废话讲了半天，还是言归正传吧，希望大家课后多进行交流，因为在我看来，技术只有不断的交流，才会有进步，闭门造车只有“S”路一条。

四．第 5 课习题

1. 什么是单片机的机器周期？什么是振荡周期？什么是指令周期？它们之间的关系是怎么样的？
2. 什么是单片机的时序？
3. 单片机有几种振荡方式？
4. 简述单片机内部时钟的产生过程。

第六课 单片机的内部结构（四）

在前一课中，我们讲述了单片机的时序和时钟，大家是不是又觉得有些头疼了，下面让我们再来做两个实验放松一下。

一．单片机 I/O 口的输出实验

1．实验程序

程序如下：

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

还是老规矩，调试，写入源代码，编译，下载，看到了什么？8 只 LED 灯都在闪烁（注意：前面的实验是让一个 LED 灯闪烁），分析一下程序：

2．程序分析

这段程序和前面的程序比较，有两处不同，第 1 条，原来是 SETB P1.0，现在改为 MOV P1，#0FFH，第 3 条，原来是 CLR P1，现在改为 MOV P1，#00H。为什么这样改了之后就变成了 8 只 LED 灯同时闪烁了？原来 P1 代表了 P1.7-P1.0 的全部，我们把它当作一个存储器单元（即一个字节），不过对一个存储器单元送数就应该用 MOV 指令了；在这里 P1（P1.7-P1.0）接的是 LED 灯（也就是负载），它起到了一个输出端的作用。那如果把 P1 改为 P0 或 P2 或 P3 行不行呢？答案是肯定的，为什么？我们稍后再谈，接着看第 2 个实验。

二．单片机 I/O 口的输入实验

1．实验程序

程序如下：

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

同样的方法把程序下载到单片机，按下第 1 个按钮，第 1 个 LED 灯亮了，按下第 2 个按钮，第 2 个 LED 灯亮了，松开按钮，相应的灯就灭了，是不是有点象工业控制中的点动控制原理。分析一下这个程序：

2．程序分析

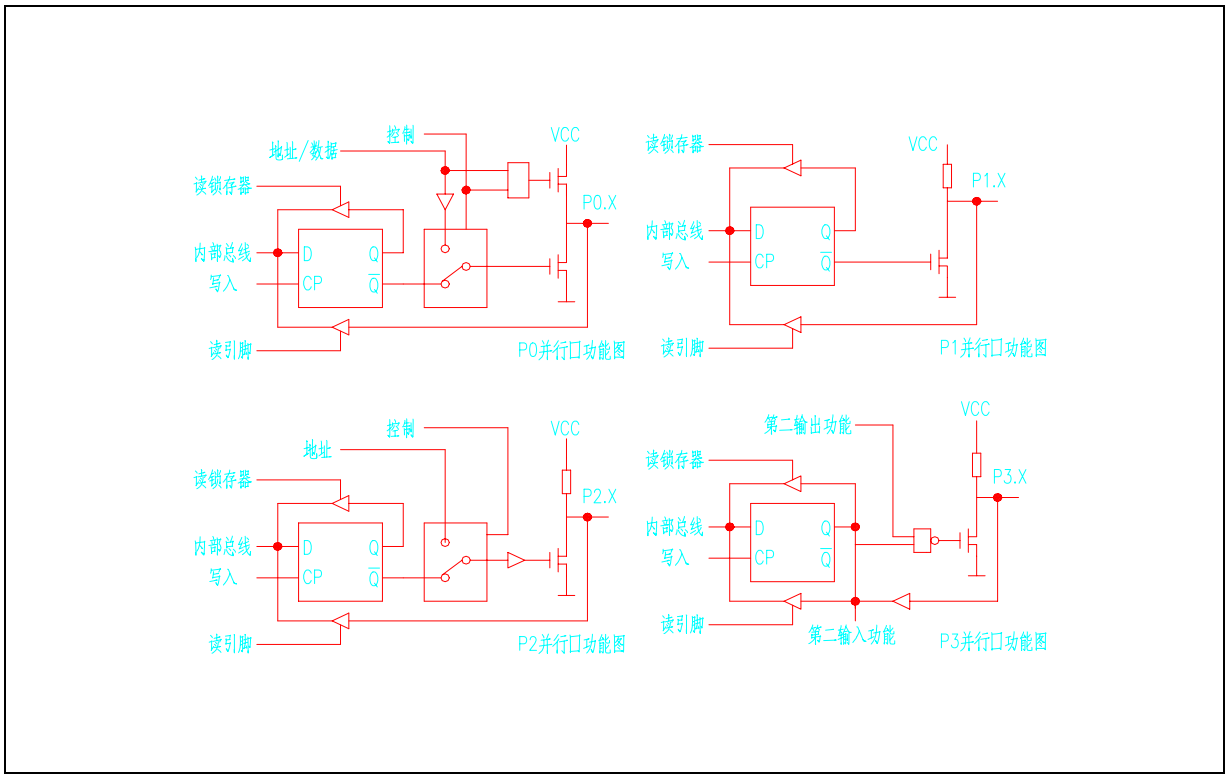
看附图的实验系统硬件接线图，有 4 个按钮分别接到了 P3.2，P3.3，P3.4，P3.5 引脚上。再来分析一下程序，第 1 条，使 P3 口（包括 P3.7-P3.0）全部为高电平（为什么 MOV P3，#0FFH 能使 P3 口全部为高电平，我们在下一课中讨论）；第 2 条 MOV A，P3；MOV 我们已经知道，是送数的意思，这条指令的意思就是把 P3 口的数送到 A 中去，A 是什么呢？我们也可以把它看成一个中间单元，就象 R7 寄存器一样，第 3 条指令就是把 A 中的数送到 P1 口去；第 4 条是循环，这些我们都已经见过，当我们按下

P3.2 所连接的按钮时，#0FFH 这个数就被送到了 A 中，通过程序又送到了 P1，使 P1.2 输出低电平，LED3 就亮了，按下 P3.3-P3.5 连接的按钮，对应的 LED4-LED6 也亮了，松开按钮，相应的 LED 灯就灭了。如果把按钮接到 P2.0-P2.7 或 P0.0-P0.7 可不可以呢？当然可以。所以在这里 P3 口又起到了一个输入端的作用。

由上面两个实验我们得出结论，凡是以 P 开头的管脚都可以用作输入输出口，在 89C51 中这 32 个管脚我们就称之为并行口。它们实际上就是特殊功能存储器 SFR（什么是特殊功能寄存器，我们后面再讲）中的四个，记作 P0，P1，P2，P3，它们都是双向通道，即既可以作为输出口，也可以作为输入口，作输出时数据可以锁存，作输入时数据可以缓冲（锁存和缓冲是什么意思？忘了，我也不告诉你，自己回去看数字电路基础，呵呵，不是我不肯讲，只是自己看可以加深印象），那么它们是怎么实现输入输出功能的呢？继续往下看。

三． 单片机并行口的结构分析

先来看看输入结构：



1．输入结构

I/O 口作为输入口时有两种工作方式，即所谓的读端口与读引脚。读端口时实际上并不从外部读入数据，而是把端口锁存器的内容读入到内部总线，经过某种运算或变换后再写回到端口锁存器。只有读端口时才真正地把外部的数据读入到内部总线，上面图中的两个三角形表示的就是输入缓冲器，CPU 将根据不同的指令，分别发出“读端口”或“读引脚”信号，以完成不同的操作，这是由硬件自动完成的，不需要我们操心。

读引脚时，也就是把端口作为外部输入线时，首先要通过外部指令把端口锁存器置“1”，然后再实行读引脚操作，否则就可能读入出错。为什么？看上面的图，如果不对端口置“1”，端口锁存器原来的状态有可能为“0”（Q 端为 0，Q^为 1）加到场效应管栅极的信号为“1”，该场效应管就导通，对地呈现低阻抗,此时即使引脚上输入的信号为“1”，也会因端口的低阻抗而使信号变低，使得外加的“1”信号读入后不一定是“1”，若先执行置“1”操作，则可以使场效应管截止，引脚信号直接加到三态缓冲器中，实现正确的读入。由于在输入操作时还必须附加一个准备动作，所以这类 I/O 口被称为“准双向”口，89C51 的 P0，P1，P2，P3 口作为输入时都是“准双向”口。接下来让我们再看另一个问题，从图中可以看出，这四个端口还有一个差别，除了 P1 口外，P0，P2，P3 口都还有其他的功能，这些功

能又是作什么用的呢？下面我们就来详细讲解这个问题：

2．端口的工作原理

（1）。P0 口

先来看 P0 口，从图中可以看到，P0 口的内部有一个 2 选 1 的选择器，它受内部信号的控制，如果在图中的位置则处在 I/O 口工作方式，此时相当于一个“准双向口”（输入时须先将口置“1”），每根口线可以独立定义为输入或输出，但是须在口线上加上拉电阻。如果将开关往另一个方向，则就是另一个功能—作为地址/数据复用总线用，此时不能逐位定义为输入/输出，它有两种用法：当作数据总线用时，输入 8 位数据；而当作地址总线用时，则输出低 8 位地址。再强调一点，当 P0 口作为地址/数据复用总线用之后，就再也不能作 I/O 口使用了。讲到这里，也许大家会感到困惑，什么叫作地址/数据复用？这其实是当单片机的并行口不够用时，需要扩展输入输出口时的一种用法，具体如何使用，这就比较复杂了，我们只能留到下册课程中再来给大家讲解，这里大家只要知道一下就可以了。了解了 P0 口，再来看 P1 口。

（2）。P1 口

同 P0 不同，P1 口只能作为 I/O 口使用，但它的内部有一个上拉电阻，所以连接外围负载时不需要外接上拉电阻，这一点 P1，P2，P3 都一样，务必请大家记住。

（3）。P2 口

P2 口作为 I/O 口线用时，与 P0 口一样，当内部开关向另一个方向时，即作地址输出时，可以输出程序存储器或外部数据存储器的高 8 位地址，并与 P0 口输出的低地址一起构成 16 位的地址线(注意和数据总线的区别，数据总线是 8 位的，很多书上都会提到 51 单片机是 8 位数据总线，16 位地址总线，但都不会解释有什么不同，看到这里你应该明白了吧)，16 位的地址可以寻址 64K 的程序存储器或外部数据存储器，为什么，下一课我们再给大家解释。这里要注意的是当 P2 口作为地址总线时，这高 8 位地址线是 8 位一起输出的，不能象 I/O 口线那样逐位定义，这和 P0 口是一样的。

（4）。P3 口

P3 口作为 I/O 口线用时，同其他的端口相同，也是“准双向口”；不同的是，P3 口的每一位都有另一种功能，也叫第二功能，各位的功能如下，它们的具体作用我们用到时再详细解释。

端口位	第二功能	注释
P3.0	RXD	串行口输入
P3.1	TXD	串行口输出
P3.2	INT0	外部中断 0
P3.3	INT1	外部中断 1
P3.4	T0	计数器 0 计数输入
P3.5	T1	计数器 1 计数输入
P3.6	WR	外部 RAM 写入选通信号
P3.7	RD	外部 RAM 读出选通信号

讲到这里，也许您会问？既然单片机的引脚有第二功能，那么 CPU 是如何来识别的呢？这是一个令许多初学者困惑的问题，几乎没有一本教科书提到过这个问题，其实单片机的第二功能是不需要人工干预的，也就是说只要 CPU 执行到相应的指令，就自动转成了第二功能。

了解了各个 I/O 口的功能和作用后，再来给大家讲解一下单片机 I/O 与外围电路的连接方法。这可是蛮重要的哦！

四．单片机 I/O 口的连接方法

当单片机的 I/O 口作输出时可以直接与外部设备连接，不过由于在实际的应用中，由于其驱动电流是有限的（DATA SHEET 上说是 20mA），所以我们常常需要通过接口电路来扩展它的驱动能力，在单片机的后向通道控制系统中，常用的功率控制器件有机械继电器、晶闸管、固态继电器等等，下面我们将以机械继电器和固态继电器的应用为例介绍其具体的使用方法。

1． 单片机与机械继电器的接口

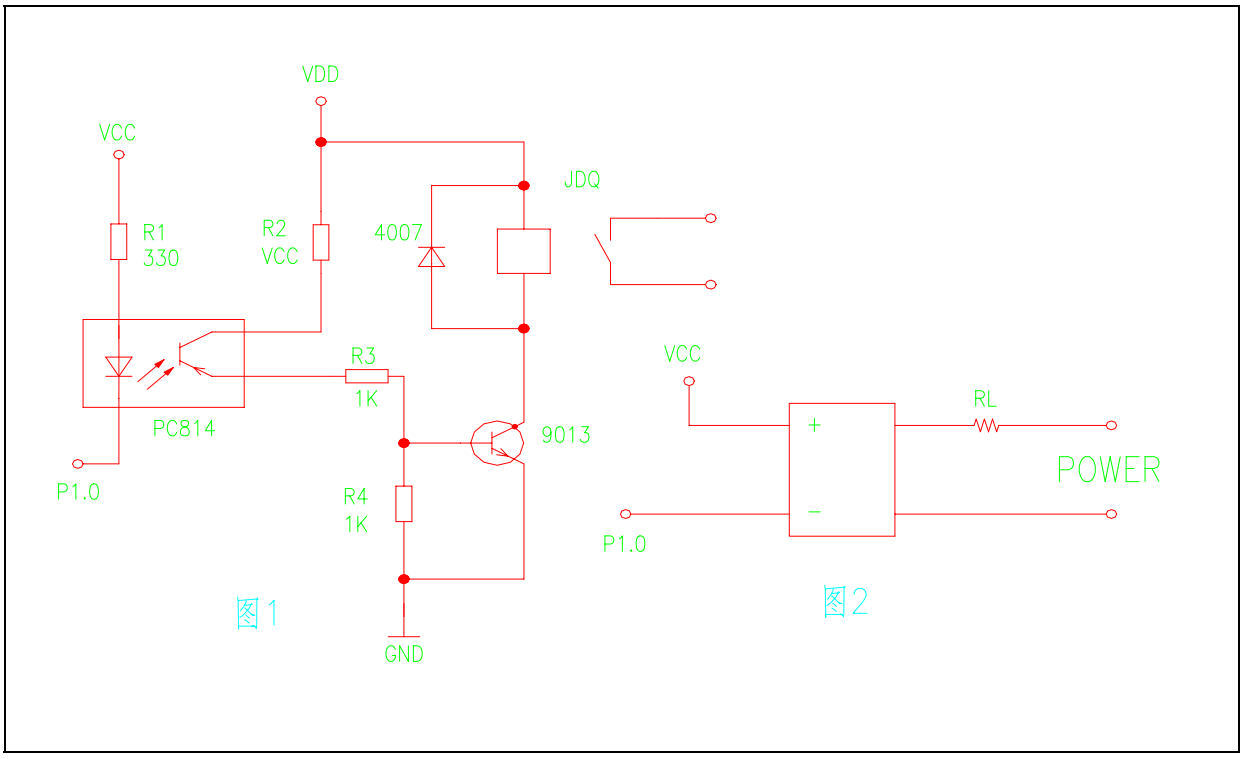
我们知道，单片机的一个 I/O 口只能灌入 20mA 的电流，所以往往不足以驱动一些功率开关（比如

稍大一点的机械继电器等），此时，就应该采用必要的扩展电路，如何来实现单片机与机械继电器的接口呢？其实很简单，我们通常采用下面的接法（如图），为了防止前向通道信号的干扰，常采用一些光电隔离器件，比如光电耦合器 4N25，PC814 等，当单片机的 P1.0 脚输出为低电平时，光藕受电导通，Q1 饱和开通，继电器吸合，负载电路接通。

另外为了防止电压间的互相干扰，继电器的工作电压 VDD 与单片机的工作电压 VCC 不要使用同一个电源，接地端也不要连在一起，即所谓的模拟地与数字地分开，驱动管的电流要大于继电器的工作电流,其他的元件就不讲了，大家自行分析一下。讲了单片机与继电器的接口，再来介绍与固态继电器的接口方法，接着往下看：

2. 单片机与固态继电器的接口

普通继电器由于开关速度慢、易跳火、易机械磨损，通常用于要求不高的场合，在某些特殊应用场合，比如防火、防爆等系统中，则应采用固态继电器。固态继电器是一种无触点的电子继电器，它的输入端只要很小的控制电流，可以与单片机的 I/O 口直接连接；输出则采用双向晶闸管控制，其输入输出间均通过内部光电耦合器隔离，可以防止信号间的干扰，是单片机接口的理想器件，随着其技术的成熟，应用的广泛，价格也已经非常的便宜，1A/250V 的目前在 10 元左右，它与单片机的连接方法如图所示，当“-”端所接的 P1.0 为低电平时，SSR 导通，负载工作。



除了以上两种连接方法外，单片机与 TTL，CMOS 管等都可以连接，具体的方法这里就不介绍了，大家可以自行找一下相关的资料。

五. 本课总结

输入和输出口（简称 I/O 口）是单片机与外部电路接口的唯一途径，四个并行口的结构是有一定区别的，如何根据系统的设计要求和产品用途来正确、灵活地使用是初学者必须掌握的基本功，我们必须好好搞清楚它的功能和用途。

六. 第 6 课习题

1. P0，P1，P2，P3 口的驱动电流分别是多少？
2. 什么是输入？什么是输出？
3. 找本数字电路的书，了解一下 D 触发器的原理。

第七课 单片机的内部结构（五）

在上一课中，我们讲到了指令 MOV P3，#0FFH 能使 P3 口全部为高电平，而在第四课中 LED 灯闪烁程序中给 R7 送数用的指令是 MOV R7，#250，那么这#250 和#0FFH 到底有什么不同？它们又代表什么意思呢？这一课就来讨论这个问题。在讲解之前，让我们先来复习一下数字电路中学过的数制概念：

一．数制

1． 十进制数（Decimal Number）

在日常生活中，我们表示数的多少用的是十进制数，即 0，1，2，3，4，5，6，7，8，9。它遵循“逢十进一，借一当十”的原则，通常我们把计数符号的个数叫做基数，十进制的基数就是十。

比如一个十进制数 5847=5*1000+8*100+4*10+7*1，它的每一个数码都有一个系数 1000，100，10，1；这个系数叫做权或位权。十进制数虽然非常符合我们的使用习惯，但计算机中却无法采用，因为计算机只能有两种状态：“0”和“1”，所以我们还得应用二进制数。

2． 二进制数（Binary Number）

二进制的基数为二，0 和 1，它遵循的是“逢二进一，借一当二”的进借位原则。也就是当某位计数到两个数时就向高位进“1”，同时本位变为“0”。

比如二进制数 1100=$1*2^3+1*2^2+0*2^1+0*2^0$，二进制数只有 0 和 1 两个数，正好代表了计算机中电路的两种工作状态，所以它在计算机中被广泛应用。下面是二进制的加法和乘法运算规则：

加法：0+0=0；1+0=0+1=1；1+1=10

乘法：0*0=0；1*0=0*1=0；1*1=1

二进制数虽然在计算机中处理很方便，但当位数较多时，就不容易记忆和书写了，所以计算机中又有了十六进制数。

3． 十六进制数（Hexadecimal Number）

十六进制也遵循两个规则，一是有十六个基数，即 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F；另一个规则是“逢十六进一，借一当十六”。

比如我们前面提到的#0FFH 就是一个十六进制数，#--我们已经明白了，它表示的是传递数的本身，“H”叫数制简码，它表示这个数是十六进制数，为什么前面我在标题后面都加了英文注释，相信大家也应该明白了吧（这里随便提一下，二进制简码 B 和十进制简码 D 通常是可以省略的，我们以后的课程中用到的数都是这样写的），那么 0FFH 这个十六进制数的表示方法是怎么样的呢？用十进制就是表示 0FFH=$F*16^1+F*16^0$，即等于 255。（大家也许会疑问？这里的“0”到哪里去了呢？原来，在单片机中，当我们用十六进制格式表示一个数时，如果高位的数字为“A-F”时，高位前面就得加上个“0”，不然，编译软件会出错，就象#0FFH。

二． 进制之间的转换

十进制有使用比较习惯的特点，二进制有易于表示和运算方便的特点，十六进制又有表示位数较多的特点，但有时我们常常要把十进制数转换成二进制数或十六进制数来处理；把二进制数逆转换成十六进制数，如何进行这种转换呢？下面就举几个例子：

1． 十进制数与非十进制数之间的转换

（1） 非十进制数转换为十进制数

具体做法是：将一个非十进制数按权展开成一个多项式，每项是该位数码与相应权值之积，把多项式按十进制的规则进行计算求和，所得的结果就是该数的十进制形式。

比如：二进制数 1011B 转换成十进制为 $1*2^3+0*2^2+1*2^1+1*2^0=8+2+1=11D$，再比如：十六进制数 FFH 转换成十进制为 255D。

（2） 十进制数转换为非十进制数

十进制数转换为非十进制数时，可将其分为整数部分和小数部分分别进行转换，最后将结果合并

为目的数。为了简单，我这里只讲整数部分的转换。这种转换叫做除基取余法，具体做法是用欲转换数制的基数去除十进制数的整数部分，第一次除所得余数为目的数的最低位，把得到的商再除以该基数，所得余数为目的数的次低位，依次类推，继续上面的过程，直至商位为 0，此时所得余数为目的数的最高位。

比如：将十进制数 53D 转换成二进制数为 53D=110101B。

```
2 └53
2 └26……1
2 └13……0
 2 └6……1
 2 └3……0
 2 └1……1
   0 ……1
```

2． 二进制数与十六进制数之间的转换

四位二进制共有 16 种组合，而这 16 种组合正好与十六进制数的 16 个基数一致，所以每 4 位二进制数对应一位十六进制数，我们只要把二进制数的整数部分自右向左每 4 位一组，最后不足 4 位的用 0 补足；小数部分自左向右每 4 位一组，最后不足 4 位的在右面补 0，再将每 4 位二进制数对应的十六进制数写出即可。相反，如果将十六进制数转换为二进制数只需将每位十六进制数写成对应的 4 位二进制数即可。

比如：将 1101011B 转换成十六进制数为 D6H，再比如：将 F0FH 转换成二进制数为 111100001111B。

十进制数	二进制数	十六进制数
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

上面的表格就是二进制数、十进制数和十六进制数之间的对应关系。

三． 立即数的写法

通过前面一小节的讲解，我们已经懂了，MOV R7，#250 和 MOV R7，#0FFH 中#250 和#0FFH 原来是十进制数 250D 和十六进制数 FFH 的区别，在单片机中，通常我们把这个数称之为立即数，那么如果我在编写指令时把立即数#0FFH 写成二进制数（即 11111111）或用十进制写法（255）是不是可以呢？当然可以，立即数既可以是二进制数，也可以是十进制数或十六进制数。讲到这里你应该明白了，为什么我们前面的实验把#0FFH 送到 P3 口会使 P3.7-P3.0 变为高电平。这里再重复一遍：那就是当用十六进制格式表示一个立即数时，如果高位的数字为“A-F”时，高位前面要加上个“0”，请大家务必记住了，这是一个常识问题，可很多书上都不提，所以很多人在做实验时往往会编译出错。

这里简单地讲了一下关于数制以及二进制、十进制和十六进制数的关系，大家可以在以后的实践

中慢慢去理解和掌握，如果您一时记不住，千万不要刻意地去死记硬背！下面让我们来讨论另一个问题：

四． 存储器的地址

什么是存储器的地址，地址和数据又有什么关系呢？这个问题往往让初学者非常的难以理解，既然单片机存储器内存放的是数据，为什么还要有地址的概念？让我们从生活中的一个例子谈起：大家都知道寄信是怎么回事吧！我们要寄一封信就必须写好信的内容，然后在信的封面写上详细地址，邮局才能按地址把它寄出去；我们给单片机送数也一样，除了要给出立即数（犹如信的内容），还必须知道这个数送达的地址（犹如信的地址或邮政编码），所以就必须给每个寄存器（即半导体存储器）都规定不同的地址，只不过在单片机中地址的编码也是用数字来表示的，那么单片机中有多少个寄存器呢？它们的地址又是如何规定的呢？

前面我们学过，单片机有两种存储器，即只读存储器 ROM 和随机存储器 RAM，它们都被规定了各自的地址，我们把它称做寻址空间。既然是空间，就必然有一个范围的概念，接下来就让我们先看看 89C51 单片机内部程序存储器的寻址范围：

1． 内部 ROM 的寻址范围

89C51 的内部有 4K 的 FLASH ROM 空间，其寻址范围为 000H-FFFH（$1516^2*1516^1*1516^0$=0-4095），这 4K 的 ROM 空间就是用来存放我们为单片机编写的程序用的，单片机执行指令时就是一条一条顺序地从 ROM 中寻找指令进行执行。 了解了 ROM 的寻址范围，让我们接着来看另一种存储器 RAM 的寻址范围：

2． 内部 RAM 的寻址范围

89C51 内部共有 128 个字节的 RAM 空间，其寻址范围为 00H-7FH（怎么算出来的，大家结合前面所学的知识自己理解一下），它被分成三个区域：第一个区域 00H-1FH 安排了 4 组工作寄存器，每组用 8 个字节，共 32 个字节，分别为 R0-R7，当然在同一时刻，只能用其中的一组工作寄存器，怎么来控制它，就要用到程序状态字 PWS 中的 RS0、RS1 两位（这我们后面再讲）；第二个区域 20H-2FH 共 16 个字节除了可以作为一般的 RAM 单元读写外，还可以对每个字节的每一位（即每一个抽屉中的每一个小盒子）进行操作，并且对这些位都规定了固定的位地址：从 20H 单元的第 0 位开始到 2FH 单元的第 7 位结束共 128 位；第三个区域就是一般的 RAM 单元，地址为 30H-7FH，共 80 个字节。

实际上，在 89C51 单片机的内部还有一个部分从 80H-FFH 是专门用于特殊功能寄存器（SFR）的，89C51 共用 21 个特殊功能寄存器（这些我们都将在下一课中讲解），它们每个也都有 8 位的，这些特殊功能寄存器的使用和前面的 128 个字节 RAM 不同，所以很多书上的解释都是 89C51 有 128 个字节的内部 RAM，实际上它们也属于内部 RAM 一部分。

为了加深印象，大家可以打开 DUG8051 软件看一下它们的内部组成。

五． 本课总结

本课主要讲述了数据与地址两个概念，其中第一部分的内容在学习数字电路时大家应该学过，我这里把与单片机有关的内容再讲解一下，目的是希望各位能掌握这些知识，因为它们对我们学习单片机是非常有用的；数据和地址是单片机中一个非常重要的概念，也是比较难以理解，在我们以后的学习中，大家会发现，我还会更加详细的讲解这方面的相关知识。

六．第 7 课习题

1．二进制、十进制、十六进制的规则分别是什么？
2．什么叫立即数？
3．单片机 RAM 的寻址空间为多少？它包括哪两个部分？
4．单片机 ROM 的寻址空间为多少？
5．把下面的立即数转换成二进制：
   100；250；100H；4AH；FFH
6．把下面的立即数转换成十进制：
   0001；0011；1111；A0H；FFH
7．把下面的立即数转换成十六进制：
   100；255；；00111100；11110101

第八课 单片机的内部结构（六）

前面我们已经讲过，R7，R6 是工作寄存器，P0，P1，P2，P3 是并行口，那么单片机中还有些什么东西？它们的结构又是怎么样的呢？这就是本课要讨论的问题。

一． 单片机的特殊功能寄存器

看第三课的单片机内部结构图，在单片机中，除了前面介绍的 RAM，ROM，P0-P3 和 CPU 外，方框内的还有许多其他的东西它们被称为特殊功能寄存器，英文简写 SFR，下表例出的就是 MCS-51 单片机中几个常用的特殊功能寄存器。这一课我们先来介绍几个：

二．几个常用的特殊功能寄存器

1.累加器 ACC

通常用 A 表示，它是一个什么东西呢？我们知道单片机在做运算时它的中间结果需要放在某个地方，这个地方就是累加器，它的名字很特殊，功能也很特殊，几乎所有的运算类指令都离不开它。

2.寄存器 B

B 寄存器在做乘法时用来存放一个乘数，在做除法时用来存放一个除数，不做乘除法时随你怎么用。

3.程序状态字 PSW

它是一个很重要的东西，里面放了 CPU 工作时的很多状态，知道它就可以了解 CPU 当前的工作状态，它有点象平时看书用的目录，我们浏览它就可以了解一本书的内容。它是一个 8 位的寄存器，用到了其中的 7 位。其格式如下：

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	0V		P

下面来逐位介绍它的功能：

（1）CY：进位标志位

MCS-51 是一种 8 位的单片机，它的运算结果只能表示到 2^8（即 0-255），但我们有时候的运算结果要超过 255，怎么办呢？就要用 CY 位。例如：79H+87H（01111001+01010111）=1 00000000，这里的“1”就进到了 CY 中去了。

（2）AC：半进位标志位

当 D3 位向 D4 位进位/借位时，AC=1，通常用于十进制调整运算中。

（3）F0：用户自定义标志位

由编程人员自行决定，什么时候用，什么时候不用。

（4） RS1、RS0：工作寄存器组选择位

RS1	RS0	工作寄存器组
0	0	0 组（00H-07H）
0	1	1 组（08H-0FH）
1	0	2 组（10H-17H）
1	1	3 组（18H-1FH）

前面讲到单片机共有四个工作寄存器组（0 组-3 组），它们就是由 RS1，RS0 来控制，这两位就在这里，它共有四种组合状态，看上面的表格：每个工作寄存器组有 8 个字节，分别记为 R0-R7，当然在某一时刻，CPU 只使用其中的一组。

假设 PSW 为“11H”（即 00010001），那么 RS1=1，RS0=0，则用到了第 2 组寄存器组（地址 10H-17H），R0-R7 即为 10H-17H，用 DBG8051 软件输入数值，看看内部 RAM 中地址为 10H-17H 中的值是不是为输入值。

（5） 0V：溢出标志位

什么时候溢出，我们讲到定时器时再研究。

（6） P：奇偶检验位

每次运算结束后若 A 中二进制数“1”的个数为奇数，则 P=1，否则 P=0。例：某运算结果是 58H（01011000），显然“1”的个数为奇数，所以 P=1。

4．DPTR（DPH，DPL）：数据指针

数据指针是一个 16 位的寄存器，我们可以用它来访问外部 RAM，也可以访问外部 ROM 中的表格，具体应用以后再讲。

5．SP：堆栈指针：

让我们先来理解一下堆栈是什么意思？你在家洗碗吗？我们洗好碗之后，是怎么放的呢？一般总是先洗的放在下面，晚洗的放在上面，然后用的时候呢，总是晚放上去的先用，先放上去的后用；如果你不洗碗不要紧，知道码头上仓库里堆的货物吗？一般也是先进去的后出来，而后进去的先出来，这种符合“先进后出，后进先出”存放规则的现象我们就把它叫做“堆栈”。（其实栈在中文中的意思就是码头）。

在单片机中，我们可以在内部 RAM 中构造出（注意☺：是可以构造）这样一个区域，这个区域存放数据的规则就符合堆栈中“先进后出，后进先出”的原则。为什么要有这样一个区域呢？存储器本身不也同样可以存放数据吗？是的，知道了存储器地址确实可以读出它里面的内容，但如果我们要读出的是一批数据，每一个数据都要给出一个地址就会很麻烦，为了简化操作就可以利用堆栈的存放方法来读取数据，具体的应用我们将在十五课中结合具体实验来讲，这里只是让大家先了解一下。那么堆栈在单片机的什么地方？也就是说把 RAM 空间的哪一块区域作为堆栈呢？这就不好定了，因为单片机是一种通用的产品，每个人的实际需要各不相同，有人需要多一些堆栈，而有人则不需要那么多堆栈，所以 INTEL 公司就干脆不分了，把分的权利让给用户（编程者），也就是说我们可以根据自己的需要来决定，所以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，看下面的图，SP 中的值等于 27H 不就相当于是一个指针指向 27H 单元吗？这就是堆栈指针的由来。

<table>
<tr><td>31H</td><td rowspan="8">←SP</td><td rowspan="8"></td><td>31H</td><td rowspan="8">第一个数据</td></tr>
<tr><td>30H</td><td>30H</td></tr>
<tr><td>29H</td><td>29H</td></tr>
<tr><td>28H</td><td>28H</td></tr>
<tr><td>27H</td><td>27H</td></tr>
<tr><td>26H</td><td>26H</td></tr>
<tr><td>25H</td><td>25H</td></tr>
<tr><td>24H</td><td>24H</td></tr>
</table>

当然在 MCS-51 单片机中，☺：指针开始所指的位置并非就是数据存放的实际位置，而是数据存放的前一个位置。例如一开始堆栈指针是指向 27H 单元的，那么第一个数据的存放位置就在 28H 单元中，而不是 27H 单元中，这一点请大家注意。

6．电源控制寄存器 PCON

单片机在以电池供电的系统中，有时为了节电，我们需要让它尽量降低电源的消耗，所以单片机就有多种的工作方式，其中一种就是低功耗方式，PCON 寄存器就是用来控制单片机进入低功耗方式的，有关这方面的知识我们将在下一课的课程中详细介绍。

三．本课总结

以上几个寄存器只是单片机中最常用的几个 SFR，其他的特殊功能寄存器，我们将在具体应用时再作详细的介绍。

四．第 8 课习题

1．累加器 A 的作用是什么？

2．什么是堆栈？堆栈存放数据的规则是什么？

3．单片机中有几组工作寄存器？它们的字节地址是什么？

4．简述 PSW 各位的作用。

第九课 单片机的工作方式

上一课中，我们提到了单片机的工作方式，单片机究竟有几种工作方式，它们又是如何工作的呢？这一课就来讨论这个问题。

一. 单片机的工作方式

单片机共有复位、程序执行、低功耗和编程与加密四种工作方式，下面分别加以介绍。

1. 复位方式

（1）为什么要复位

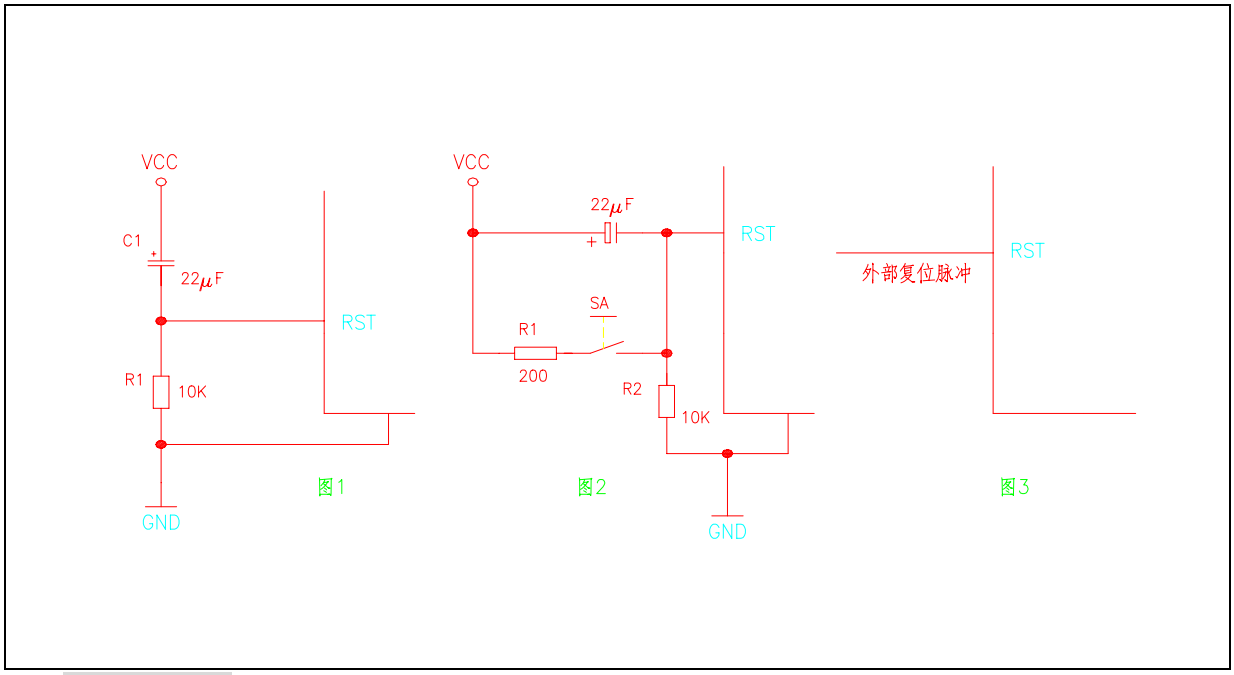
大家知道，单片机执行程序时总是从地址 0000H 开始的，所以在进入系统时必须对 CPU 进行复位，也叫初始化；另外由于程序运行中的错误或操作失误使系统处于死锁状态时，为了摆脱这种状态，也需要进行复位，就象电脑死机了要重新启动一样。

（2） 复位的原理

单片机复位的方法其实很简单，只要在 RST 引脚（9 脚）上加一个持续时间为 24 个振荡周期（即两个机器周期）的高电平就可以了。如果晶振为 12M，计算一下这个持续脉冲需要多长时间？

（3） 如何进行复位

复位操作有上电自动复位、按键复位和外部脉冲复位 3 种方法，它们的电路分别如下：



上电自动复位是通过外部复位电路的电容充电来实现的，看图 1，当电源刚接通时电容 C 对下拉电阻开始充电，由于电容两边的电压不能突变，所以 RTS 端维持高电平，只要这个充电时间不超过 1ms，一般都就可以实现对单片机的自动上电复位，即接通电源就完成了系统的初始化，在实际的工程应用中，如果没有特殊要求，一般都采用这种复位方式；按键复位的电路如图 2 所示，它其实就是在上电复位的基础上加了 R1 和 SA，这种电路一般用在需要经常复位的系统中；外部脉冲复位的电路如图 3 所示，外部复位通常用于要求比较高的系统，比如希望系统死锁后能自动复位。外部复位是由专门的集成电路来实现的，也就是我们通常俗称的“看门狗”电路，这种电路有很多，它们不但能完成对单片机的自动复位功能，而且还有管理电源、用作外部存储器等功能，比如 X25045,MAX813L 等等就是比较常用的此类芯片，关于这方面的内容我们将留到下册的教程中再来给大家详细讲解。

现在让我们先来看看单片机复位后，它的内部会有些什么变化呢？看下面的表：

（4）复位后的状态

这就是单片机复位后内部系统的状态，上面的有些符号我们暂时还看不懂，不过没关系，等以后学到了相关的知识后您自然就会明白了。

单片机的初始化状态	
寄存器	复位时的内容
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0-P3	FFH
TMOD	00000000B
TCON	0X000000B
TL0	00H
TH0	00H
TL1	00H
TH1	00H
SCON	00H
SBUF	不定
PCON	0XXX0000B

2．程序执行方式

程序执行是单片机的基本工作方式，由于复位后 PC=0000，所以程序就从地址 0000H 开始执行，此时单片机就根据指令的要求完成一系列的操作控制，比如前面讲的让 LED 灯闪烁起来，不过在实际使用中，程序并不会从 0000H 开始执行，而总是安排一条跳转指令，比如 LJMP　START，为什么要这样安排，我们讲到中断时再来解释。

3．低功耗操作方式

在以电池供电的系统中，有时为了降低电池的功耗，在程序不运行时就要采用低功耗方式，低功耗方式有两种—待机方式和掉电方式。

低功耗方式是由电源控制寄存器 PCON（上一课我们提到过的）来控制的。电源控制寄存器是一个逐位定义的 8 位寄存器，其格式如下，

MSB							SB
SMOD	--	--	--	GF1	GF0	PD	IDL

其中：SMOD 为波特率倍增位，在串行通讯时用；GF1 为通用标志位 1；GF0 为通用标志位 0；PD 为掉电方式位，PD=1，进入掉电方式；IDL 为待机方式位，IDL=1，进入待机方式。也就是说只要执行一条指令让 PD 位或 IDL 位为 1 就可以了。那么单片机是如何进入或退出掉电工作方式和待机工作方式的。我们来介绍一下：

（1） 待机方式

① 进入待机方式

当使用指令使 PCON 寄存器的 IDL=1，则进入待机工作方式。此时 CPU 停止工作，但时钟信号仍提供给 RAM，定时器，中断系统和串行口；同时堆栈指针 SP，程序计数器 PC，程序状态字 PSW，累加器 ACC 以及全部的通用寄存器都被冻结起来；单片机的消耗电流从 24mA 降为 3.7mA，这样就可以节省电源的消耗。

② 退出待机方式

退出待机方式可以采用引入中断的方法，在中断程序中安排一条 RETI 的指令就可以了，什么是中断，我们现在还不知道，当然这没关系。其实待机方式和我们使用电脑时的睡眠方式有异曲同工之妙。

（2）掉电方式

①进入待机方式

当使用指令使 PCON 寄存器的 PD=1，则进入掉电工作方式，此时单片机的一切工作都停止，只有内部 RAM 的数据被保持下来；掉电方式下电源可以降到 2V，耗电仅 50uA。此时就相当于把显示器和硬盘也关闭了。

② 退出待机方式

退出掉电工作方式的唯一方法是复位，不过应在电源电压恢复到正常值后再进行复位，复位时间要大于 1mS，在进入掉电方式前，电源电压是不能降下来的，因此可靠的单片机电路最好要有电源检测电路。显然掉电方式和待机方式是两种不同的低功耗工作方式，前者可以在无外部事件触发时降低电源的消耗，而后者则在程序停止运行时才使用。

关于单片机的低功耗的方式就简单的讲这些，更详细的内容也留到下册再讲解。

4. 编程和加密方式

单片机的编程与加密是由专门的设备来完成的，这种设备称为编程器或烧录器，类似的产品有很多，功能也不尽相同，如果您有兴趣，我将在以后给您介绍一款 51 系列单片机编程器的自制方法。

这里给大家简单介绍一下单片机的加密，加密是为了保护编程者的劳动成果而设计的一种工作方式，不过有矛必然有盾，现在的高手实在是很多，听说即使用 OTP 特种加密方式，也能解密，不过能加密总比不加密的好，所以大家在编程时应尽量采用加密功能。

二. 本课总结

这一课我们讲述了单片机的工作方式，对于初学者来说除了复位方式外，其他的只要稍微有点了解就可以了。

三. 第 9 课习题

1. 单片机有几种工作方式？
2. 为什么要进行复位？复位后的状态是什么？
3. 如何对单片机进行复位？
4. 找一套编程器的软件自己先熟悉一下。

第十课 单片机的寻址

这一课让我们来了解一下单片机的寻址方式，这对大家掌握指令会有很大的帮助，什么是单片机的寻址？单片机有几种寻址方式？请往下看：

我们已经知道，单片机的工作过程就是一条一条地从 ROM 存储器中取出指令然后执行相关的操作，那么一条指令究竟有哪几部分组成？它又包括哪些内容？一般来说一条指令总是有操作码字段和操作数字段两部分组成，看下面两条指令，MOV R7，#250；MOV P1，#0FFH，这是我们以前学过的指令，在这两条指令中 MOV 就是操作码字段，R7 和 P1 就是操作数地址字段，而#0FFH 我们称为常数(也就是立即数)，单片机执行指令时就根据指令中给出的地址寻找实际的操作数，不能理解，没关系，继续往下看。

一．单片机的寻址

先来看下面的实验：

程序一	程序二
MAIN: SETB P1.0 ；（1）	MAIN: SETB P1.0 ；(1)
	MOV 30H，#255 ；
LCALL DELAY ；（2）	LCALL DELAY ；(2)
CLR P1.0 ；（3）	CLR P1.0 ；（3）
	MOV 30H,#200 ；
LCALL DELAY ；（4）	LCALL DELAY ；（4）
AJMP MAIN ；（5）	AJMP MAIN ；（5）
DELAY: MOV R7，#250 ；（6）	DELAY: MOV R7，30H ；（6）
D1: MOV R6，#250 ；（7）	D1: MOV R6，#250 ；（7）
D2: DJNZ R6，D2 ；（8）	D2: DJNZ R6，D2 ；（8）
DJNZ R7，D1 ；（9）	DJNZ R7，D1 ；（9）
RET ；（10）	RET ；（10）
END ；（11）	END ；（11）

程序一就是我们以前做过的 LED 灯闪烁的实验，我们已经知道每次调用延时程序的时间都是相同的（125mS)，如果现在提出这样的要求：灯亮后延时时间为 125mS 灯灭，灯灭后又延时 100mS 秒灯亮，如此循环，这样的程序还能满足要求吗？显然不能，怎么办？我们可以把它改成程序二，也就是先把一个数送入 30H，在子程序中 R7 中的值并不固定，而是根据 30H 单元中传过来的数来确定，这样就可以满足要求，大家自行分析一下这个程序。

从这里我们可以得出结论，在数据传递中要找到被传递的数，很多时候，这个数并不能直接给出，而是需要变化，这就引出了一个概念：如何寻找操作数，我们把寻找操作数所在单元地址的过程称之为寻址。在实验一中，我们直接使用数所在单元的地址找到了操作数，所以称之为直接寻址。而在实验二中，我们是把数先放在工作寄存器 30H 中，再把 30H 中的数送到 R7（看实验二的第 6 条指令），这种方式则称之为寄存器寻址，讲到这里，大家有没有看出来，这里的 30H 前面是没有#的，而象 MOV R7，#250 这样的指令在 250 前是有#的，为什么？我前面提到过，大家好好回顾一下。

接下来提一个问题：我们知道，工作寄存器就是内存单元的一部份，如果我们选择工作寄存器组 0，则 R0 就是 RAM 的 00H 单元，那么这样一来，MOV A，00H，和 MOV A，R0 不就没什么区别了吗？为什么要加以区分呢？的确，这两条指令执行的结果是完全相同的，都是将 00H 单元中的内容送到 A 中去，但是执行的过程不同，执行第 1 条指令需要 2 个周期；而执行第 2 条则只需要 1 个周期，第 1 条指令变成最终的目标码要两个字节（E5H 00H），而第 2 条则只要一个字节（E8h）就可以了。

也许有朋友会问，不就差了一个周期吗，为什么怎么斤斤计较！如果是 12M 晶振的话，也就 1 个微秒，一个字节又能有多少呢？当然如果这条指令只执行一次，也许无所谓，但一条指令如果执行上 1000 次，就是 1 毫秒，如果要执行 1000000 次，就是 1S 的差别，这就很可观了，单片机要做的就是实时控制，所以必须如此“斤斤计较”。

接下来再看另一个问题，现在我们已经知道，寻找操作数可以通过直接给的方式（立即寻址）和直接给出数所在单元地址的方式（叫间接寻址），有这两中方式这就够了吗？看下面的问题，要求从 30H 单元开始，取 20 个数，分别送入累加器 A 中。就我们目前掌握的办法，要从 30H 单元取数，就用 MOV A，30H，那么下一个数呢？是 31H 单元的，怎么取呢？还是只能用 MOV A，31H，那么 20 个数，不是得 20 条指令才能写完吗？这里只有 20 个数，如果要送 200 个或 2000 个数，那岂不要写上 200 条或 2000 条命令?这未免也太笨了吧。为什么会出现这样的状况？因为我们现在只会把地址写在指令中，所以就没办法了，如果我们不是把地址直接写在指令中，而是把地址放在另外一个寄存器单元中，根据这个寄存器单元中的数值决定该到哪个单元中取数那就没问题了。比如，当前这个寄存器中的值是 30H，那么就到 30H 单元中去取，如果是 31H 就到 31H 单元中去取，就可以解决这个问题了。怎么做呢？既然看的是寄存器中的值，那么我们就可以通过一定的方法让这里面的值发生变化，比如取完一个数后，将这个寄存器单元中的值加 1，还是执行同一条指令，可是取数的对象却不一样了。看下面的例子：

```
        MOV  R7，#20  ；（1）
        MOV  R0，#30H ；（2）
LOOP：  MOV  A，@R0   ；（3）
        INC  R0       ；（4）
        DJNZ  R7,LOOP ；（5）
```

这个例子中的大部份指令我们是能看懂的，第 1 条，是将立即数 20 送到 R7 单元中，执行完后 R7 中的值应当是 20；第 2 条是将立即数 30H 送入 R0 工作寄存器中，所以执行完后，R0 工作寄存器中的值是 30H；第 3 条，是看一下 R0 单元中是什么值，把这个值作为地址，取这个地址单元的内容送入 A 中，此时，执行这条指令的结果就相当于执行 MOV A，30H；第 4 条，没学过，就是把 R0 中的值加 1，因此执行完后，R0 中的值就是 31H 了；第 5 条，学过，将 R7 中的值减 1，看是否等于“0”，不等于“0”，则转到标号 LOOP 处继续执行，因此，执行完这句后，将转去执行 MOV A，@R0 这一条，此时相当于执行了 MOV A，31H（因为此时的 R0 中的值已是 31H 了）；如此，直到 R7 中的值逐次相减等于“0”，也就是循环 20 次为止，就实现了我们的要求：从 30H 单元开始将 20 个数据送入 A 中。这是另一种寻找数据的方法，由于数据是间接被找到的，所以把这种寻址方式称之为寄存器间址寻址。（注意☺，在寄存器间址寻址中，只能用 R0 或 R1 来存放等待寻找的数据，当然还可以用 DPTR 或 PC 访问外部存储器的数据，只不过在这里我们讨论的是寄存器间接寻址，所以这两个就不在讨论范围之内）。

除了以上几种寻址方法外，单片机还有变址寻址，相对寻址和位寻址共七种寻址方式。这些您暂时可以不去深究它，我们以后会结合具体的实验再来详细介绍，这里只是为了归类，所以才把它们例举在一起。下面举几个例子说明一下：

二．寻址方式举例

1．直接寻址

直接寻址时，指令中的地址码部分直接给出了操作数的有效地址。

例如：MOV A，4FH ；A←（4FH）

可用于直接寻址的空间有内部 RAM 的低 128 字节（包括其中的位寻址区与特殊功能寄存器）。

2．寄存器直接寻址

寄存器寻址时，指令中地址码给出的是某一通用寄存器的编号，寄存器的内容为操作数。

例如：MOV A，R7 ；A←（R7）

可用于寄存器寻址的空间有 R0-R7，ACC，CY（位），DPTR，B 等。

3．寄存器间接寻址

寄存器间接寻址时，指令中给出的寄存器的内容为操作数的地址，而不是操作数本身。

例如：MOV A，@R0 ；A←[(R1)]

可用于寄存器间接寻址的空间只能是 R0 和 R1，用 DPTR 或 PC 可间接寻址 64K 字节外部的 RAM 或 ROM.

4．立即寻址

立即寻址时，指令中地址码部分给出的就是操作数本身。

例如：MOV A，#0FFH ；A←0FFH

可用于立即寻址的空间有内部

5．变址寻址

变址寻址时，用指定变址寄存器的内容与指令中给出的偏移量相加 DPTR 所得的结果作为操作数的地址。

例如：MOVC A，@A+DPTR ；A← [(A)+(DPTR)]。

无论用 DPTR 或 PC 作为基准指针，变址寻址只适用于程序存储器（即 ROM），通常用于读取数据表。

6．相对寻址

相对寻址时，由程序计数器 PC 提供的基准地址与指令中提供的偏移量 rel 相加，得到操作数的地址。

例如：SJMP rel ；PC←（PC）+2+rel

7．位寻址

位寻址时，操作数是二进制数的某一位，其位地址出现在指令中。

例如：SETB bit ；（bit）←1

可用于位寻址的空间有内部 RAM 的可位寻址区和 SFR（特殊功能寄存器）中的字节地址可以被 8 整除（即地址以 0、8、F 结尾）的寄存器空间。

三． 本课总结

这一课主要讲述了单片机的寻址方式，寻址是单片机中一个非常重要的概念，单片机执行指令实际上就是到不同的地址空间寻找操作数的过程，请大家务必搞清寻址的概念和寻址的方法。

到本课为止，我们已经连续讲了很多单片机的基本概念，可能有些朋友会觉得很难，我这里可以告诉大家，如果您有这种感觉，那绝对是好事，因为学习使用单片机本来就不是一朝一夕的事，需要长期的结累和实践，只有持之以恒，才能取得最后的胜利！其实世上很多事都是如此。

不过话又说回来，当碰到一时无法理解的概念和知识时，如何来搞懂它呢？对于单片机学习来说，我可以给大家介绍一个简单的好方法：那就是先放着再说，继续往下学，等学会了后面的概念时你会突然发现有很多原来的不懂的东西会变得非常的简单。试试看，不要停留，继续往下看！

四． 第 10 课习题

1．什么是单片机的寻址？单片机有几种寻址方式？

2．单片机的指令有几部分组成？

3．直接寻址和间接寻址的区别在哪里？

4．写一段从 0AH 单元开始，把 20 个数送入 A 中的程序。

第十一课 单片机的指令（一）

指令就是编程者给单片机下的命令，也就是我们平常所说的单片机软件，前面我们已经陆续地讲到了一些指令，但还远远不够，从这一课开始就要全面的讲解指令了，希望大家多动手实验，巩固所学的知识，说实在的，其实单片机并不难学。

为了让大家比较容易记忆，按照常规分类，我把单片机的 111 条指令分成了五类—即数据传递类指令、算术运算类指令、逻辑运算类指令、控制转移类指令和位操作指令。这一课先来看数据传递类指令：

一．数据传递类指令

数据传递类指令是单片机中用的最多的指令，在 51 系列单片机的 111 条指令中共有 28 条是数据传递类指令，前面我们已经学到了几条，比如 MOV　R7，#250；MOV　A，R6 等，那么它们是怎么分类的呢？请往下看：

1．以累加器为目的操作数的指令

（1）MOV　A，Rn

（2）MOV　Rn，A

（3）MOV　A，direct

（4）MOV　A，@Ri

（5）MOV　A，#data

指令(1)把 Rn 中的数送入累加器 A，Rn 代表工作寄存器 R0-R7(以后我们只要写到 Rn 都代表 R0-R7，这一点请大家记住了)；指令（2）则相反，把累加器 A 中的数送入工作寄存器中；指令（3）是把直接地址中的数送入累加器 A 中，driect 就代表直接地址（以后也相同）；而指令（4）就是上一课我们讲的寄存器间接寻址，什么意思？这里再重复一遍，就是看一下工作寄存器中是什么值，把这个值作为地址，把这个地址中的数送入累加器 A 中，Ri 代表什么意思呢？就是工作寄存器 R0 或者 R1（以后如果写 Ri 都代表 R0 或 R1）；第（5）条指令就是把立即数（也叫常数）直接送入累加器 A 中，很显然 data 就代表立即数（以后也相同），其实这个我们以前提到过，加#的数就代表送入的是这个数的本身。

接下来举几个实例加以说明，大家可以用 DUG8051 这个软件验证一下：

A．MOV　R7，#20H；
　　MOV　A，R7 。　　将工作寄存器 R7 中的值 20H 送入 A，R7 中的值保持不变。

B．MOV　A，#250；
　　MOV　R7，A　。　　将 A 中的值 250 送入工作寄存器 R7，A 中的值保持不变。

C．MOV　30H，#20H；
　　MOV　A,30H 。　　将内存 30H 单元中的值 20H 送入 A，30H 单元中的值保持不变。

D．MOV　20H，#250；
　　MOV　R0，#20H；
　　MOV　A,@R0 ；　　先看 R0 中是什么值，把这个值作为地址，并将这个地址单元中的值送入 A 中。执行命令前 R0 中的值为 20H，则是将 20H 单元中的值 250 送入 A 中，最终送到 A 中的值是 250。

E．MOV　A,#20H ；　　将立即数 20H 送入 A 中，执行完本条指令后，A 中的值是 20H。

这里有一点要解释一下，在实例 D 中，MOV　20H，#250 中的 20H 是一个地址，而 MOV　@R0,#20H 中的 20H 却是一个地址，为什么，大家结合前面的知识好好想想，如果不明白，说明你对地址和数据的概念还没有搞清楚，呵呵，那就有点麻烦了*^*

2．以寄存器 Rn 为目的操作数的指令

（1）MOV　Rn，A

（2）MOV　Rn，direct

（3）MOV　Rn，#data

举几个实例大家自行分析一下，这个应该不难吧！

A．MOV　R7，A　　；

B．MOV　R7，30H ；

C．MOV　R7，#20 ；

这组指令功能是把源地址单元中的内容送入工作寄存器，源操作数不变。

3．以直接地址为目的操作数的指令

（1）　MOV　direct，A

例如：MOV　30H，A（将累加器 A 中的数送入内存单元 30H）

（2）　MOV　direct，Rn

例如：MOV　30H，R7（将寄存器 R7 中的数送入内存单元 30H）

（3）　MOV　direct，direct

例如：MOV　30H，20H（将内存单元 20H 中的数送入内存单元 30H）

（4）　MOV　direct，@Ri

例如：MOV　30H，@R0（看一下 R0 中是什么值，把这个值作为地址，并将这个地址单元中的值送入 30H 中。如执行指令前 R0 中的值为 20H，则是将 20H 单元中的值送入 30H 中）。

（5）　MOV　direct，#data

例如：MOV　30H，#20H（将立即数 20H 送入内存单元 30H，注意：和第三条指令的区别（MOV　30H，20H），这里的 20H 是一个 16 进制数，为什么？结合前面的知识自己想一下）

4．以间接地址为目的操作数的指令

（1）　MOV　@Ri,A

（2）　MOV　@Ri,direct

（3）　MOV　@Ri,#data16

这三条指令就不介绍了，大家自行分析一下，不过有一点希望大家记住，在这里 Ri 只能用工作寄存器 R0 或者 R1。

5．十六位数的传递指令

MOV　DPTR，#data16

指令说明：这是 51 单片机中唯一的一条 16 位立即数传递指令，大家知道 51 系列单片机是一种 8 位单片机，8 位单片机所能表示的最大数只能是 2^8=0-255。讲到这里大家应该明白了，为什么我们前面的实验中立即数不能大于 255。如果现在有个数是 1234H，我们要把它送入 DPTR，该怎么办呢？当然有办法，INTEL 公司已经把 DPTR 分成了两个寄存器，DPH 和 DPL（看一下前面的特殊功能寄存器介绍），我们只要把 12H(高 8 位)送入 DPH，把 34H(低 8 位)送入 DPL 中去就可以了，所以执行指令 MOV　DPTR，#1234H 和执行指令 MOV　DPH，#12H（1）；　MOV　DPL，#34H（2）；是一样的。

二．指令练习

请写出下列每条指令的执行结果，并用 DBG8051 软件进行验证，看结果是否正确。

1.MOV　12H,#34H

2.MOV　R0,#23H

3.MOV　R7,#22H

4.MOV　R1,12H

5.MOV　A,@R0

6.MOV　34H,@R1

7.MOV　45H,34H

8.MOV　12H,DPH

9.MOV　R0,DPL

三．本课总结

本课主要讲述了以累加器为目的操作数的指令，以寄存器为目的操作数的指令，以直接地址为目的操作数的指令，以间接地址为目的操作数的指令和一个十六位数的数据传递类指令，这些指令有的我们前面已经学到过，希望大家用 DBG8051 软件多练习，以加强对数据及指令的认识。

四. 第 11 课习题

1. Rn，Ri，direct，data 分别代表什么？举例说明。
2. 地址 30H 和数据 30H 有什么区别？
3. DPTR 是什么？#data16 代表什么？

第十二课 单片机的指令（二）

接着上一课的数据传递类指令，这一课继续讲解其他的数据类指令。提示：下面的内容我们下册中才会用到，这里只是为了把数据传递类指令讲完，才提前把它们讲一下，您不知道也没关系。

一．数据传递类指令

6． 累加器 A 与片外 RAM 之间的数据传递类指令

什么是片外 RAM（即片外数据存储器）呢？单片机不是有内部 RAM 吗？为什么还要片外 RAM 呢？难道单片机的内部 RAM 还不够用吗？的确如此，当单片机的内部 RAM 不够时，我们就要扩充 RAM 空间。那么单片机能扩充多少的外部 RAM 空间呢？89C51 单片机的片外 RAM 可以扩展到 64K，即从 0000H-FFFFH，那么它是怎样和累加器 A 进行数据传递的？（随便说一下，与外部 RAM 进行数据传递必须通过累加器 A）它们之间的传递指令共有以下四条：

（1） MOVX A，@Ri

（2） MOVX @Ri，A

（3） MOVX A，@DPTR

（4） MOVX @DPTR，A

指令说明：

A．在 51 系列单片机中，所有要送入或读出外部 RAM 的数据必须先送到 A 中去，由此我们可以看出内外部 RAM 的区别：内部 RAM 间可以直接进行数据的传递，而外部 RAM 则不行。比如，要将外部 RAM 中某一单元（假设为 100H 单元的数据）送入另一个单元（假设为 200H 单元），就必须先将 100H 单元中的内容读入 A 中，然后再送到 200H 单元中去。在这里有一个问题：CPU 是如何区分内、外部 RAM 的？大家看这里的四条指令，其操作码都是 MOVX，而内部 RAM 的操作码则是 MOV，CPU 就是根据不同的指令来自动区分读写内、外部 RAM 的。

B．要读出或写入外部的 RAM，当然还必须知道外部 RAM 的地址，在后两条指令中，地址是被直接放在 DPTR 中的；而前两条指令由于 Ri（即 R0 或 R1）只是一个 8 位的工作寄存器，所以只能提供低 8 位的地址。所以不同的应用场合就要使用不同的读写指令。

C．使用时应当首先将要读出或写入的地址送入 DPTR 或 Ri 中，然后再用读写指令。

举例说明：将外部 RAM 中 100H 单元中的内容送入外部 RAM 中 200H 单元中。

```
MOV   DPTR，#100H
MOVX  A，@DPTR
MOV   DPTR,#200H
MOVX  @DPTR,A
```

7． 累加器 A 与片外 ROM 之间的数据传递类指令

MOVC A，A+@DPTR

前一小节讲了累加器 A 与外部 RAM 之间的数据传递类指令，接下来再来讲讲片外 ROM 与累加器 A 之间的数据传递类指令。在讲解之前，先来了解一下内部 ROM 和外部 ROM 的组成，89C51 的内部有 4K 的 FLASH ROM 空间，其地址为 000H-FFFH，片外可以扩展到 64K（OOOOH-FFFFH），在这 64K 的 ROM 空间中，有 4K 字节的地址是片内和片外公用的（即 000H-FFFH），而 1000H-FFFFH 的空间是片外 ROM 专用的。讲到这里大家就会问：既然有 4K 的地址是公用的，那么 CPU 是如何区分的呢？不知大家是否还记得，在第二讲单片机的硬件电路中，有一个引脚 EA（即 31 脚），当 EA=1，CPU 从片内 ROM 的 4K 字节中取指令，如果地址超过了 4K（FFFH），单片机就自动转向片外 ROM 取指令，大家注意☺：这个过程是自动完成的，不需要人工干预；而当 EA=0 时，CPU 只从片外 ROM 取指令。所以在实际的应用中，如果只使用内部 ROM，一般总是把 EA 脚接电源，我们的实验板就是这么做的。

讲到这里，细心的朋友还会有一个问题当使用外部 ROM 和外部 RAM 时，它们的寻址范围都是

0000H-FFFFH，也就是说它们在地址上是重叠的，那么 CPU 在读取指令时又是如何来区分当前是从 ROM 取指令还是从 RAM 取指令呢？请大家来看第二课的 89C51 单片机硬件电路图，29 脚是 PSEN，当我们置位 PSEN 时（即 PSEN=1），CPU 就读取外部 ROM 指令；而要从外部 RAM 读取指令时就要置位 WR（即 16 脚）或 RD（即 17 脚），这样即使 ROM 地址和 RAM 地址是重叠的，也不会出现混乱。这里又有一个问题了，16 脚和 17 脚不是并行口 P3.6 和 P3.7 吗？如果我们把它当作第二功能 WR 和 RD 来使用，CPU 又是如何来区分的呢？这个问题我们前面已经讲过了，这里再重复一遍：单片机引脚的第二功能是不需要人工干预的，也就是说只要 CPU 执行到相应的指令，就自动转成了第二功能。

了解了片外 ROM 的读取指令原理，再来看片外 ROM 与累加器 A 之间的数据传递指令（注意☺：ROM 只能读取指令，而不能写入数据，这一点和 RAM 是不同的，不需要解释了吧）。

MOVC　A，A+@DPTR

指令说明：

A. 本条指令是将 ROM 中的数送入 A 中。通常称其为查表指令，我们常用此指令来查一个已做好在 ROM 中的表格。

B. 此条指令引出一个新的寻址方法：变址寻址。本指令是要在 ROM 的一个地址单元中找出数据，显然必须知道这个单元的地址，这个单元的地址是这样确定的：在执行本指令前 DPTR 中有一个数，A 中也有一个数，执行指令时，将 A 和 DPTR 中的数加起来，就成为要查找的数的单元地址。把查找到的结果放在 A 中，因此，本条指令执行前后，A 中的值不一定相同。

举个例子：有一个数在 R0 中，要求用查表的方法确定它的平方值（此数取值范围是 0-5）

```
MOV   DPTR，#TAB
MOV   A，R0
MOVC   A，@A+DPTR
.
.
TAB: DB 0,1,4,9,16,25
```

假设 R0 中的值为“2”，送入 A 中，而 DPTR 中的值则为“TAB”，则最终确定的 ROM 单元的地址就是“TAB+2”，也就是到“TAB+2”这个单元中去取数，取到的是“4”(DB 后面的第三个数)。其它数据也可以次类推。从这里可以看出，我们使用了标号（象 TAB 等）来代替具体的 ROM 单元地址，事实上，标号的真实含义就是地址的数值，在这里它就代表了 TAB+0，TAB+1，……TAB+25 这几个数据在 ROM 中的存放位置；而我们以前学过的如 LCALL　DELAY 指令，DELAY 代表的是以 DELAY 为标号的那段程序在 ROM 中存放的起始地址，CPU 就是根据这个起始地址才找到指令的，无法理解是吗？没关系，让我们先来看几个符号的含义就会明白了。

二．单片机的伪指令

我们前面简单提到过，END 是伪指令，那么到底什么是伪指令？它在单片机中有什么作用呢？接下来我们就来讨论这个问题，伪指令是单片机中用来给寄存器定义或者赋值的特殊指令，为什么要用伪指令呢？让我们先来看下面的实验：

1．DB—定义字节伪指令

它的功能是从程序存储器 ROM 单元的某个地址开始，存入一组规定好的 8 位二进制常数。

例如：ORG　2000H；

TAB：DB　45，48H，10；34H；

解释一下，ORG—程序开始地址伪指令，什么意思呢？就是说本条指令的下一条从该地址开始存放数据，比如上面的指令经汇编后，将从地址 2000H 开始给若干个 ROM 单元赋值，即（2000H）=45，（2001H）=48H，（2002H）=0AH，（2003H）=34H。讲到这里，有的人会问：在这些指令中，我直接用 MOV　2000H，45H；MOV　2001H，48H……不就得了，干吗要用 DB 指令呢？是的，从理论上讲，两者的效果是一样的，只是因为我们现在的程序都很短，单片机不可能只做这些简单的工作，当程序比较长时，这些指令的意义就不一样了。

除了刚刚提到的 ORG 和 DB 伪指令外，单片机中还有那些伪指令呢？下面简单讲解一下：

2．DW—定义字伪指令

在单片机中，一个字由两个字节组成，也就是说，如果一个字节可以表示一个 8 位数的话，那么一个字就可以表示一个十六位的数（关于这方面的问题我们留到下册中再来讨论，这里就不讲了，以免增加大家的学习难度），如此一来，这条伪指令的功能也就清楚了，就是从指定的 ROM 单元开始，定义若干个 16 位常数；上一课我们已经讲过，51 系列单片机要存放一个 16 位的常数就必须把这个数分成两个 8 位数据来存放，通常我们把一个 16 位数的高 8 位放入低地址，而把低 8 位放入高地址（注意☺：这两个地址必须是紧挨着的）。

例如：ORG　3000H；

　　ABC：DW　2345H，0A859H；

程序经汇编后，（3000H）=23H，（3001H）=45H，（3002H）=A8，（3003H）=59H。

3．DS—保留空间伪指令

它的功能是从指定的地址开始，保留若干个字节的 ROM 空间留作它用。

例如：ORG　2000H；

　　ABC：DS0　8H；

　　LOOP：MOV　A，30H；

汇编以后，从 2000H 开始，将保留 8 个 ROM 单元留作它用，那么以 LOOP 为标号的指令就存放在 2008H 单元中。这里有一点请大家注意☺：这几条伪指令都只能对程序存储器（ROM）起作用，而不能用它们来对数据存储器（RAM）进行赋值或做其他的工作。至于它们到底有什么作用，我们什么时候才需要用到它们？我们将在下册的实验中再作讲解。现在让我们通过一段程序来重点解释一下查表程序的使用方法，这可是一定要学会的哦。

```
例如：MOV   DPTR，#100H   ；
      MOV   A，R0         ；
      MOVC  A，@A+DPTR    ；
      .
      .
      ORG   0100H         ；
      DB 0,1,4,9,16,25 ；
```

如果 R0 中的值为“2”，则最终地址为“100H+2”即“102H”，到 102H 单元中找到的是“4”。这个可以看懂了吧。那么 103H 中的数是多少呢？104H 呢？大家思考一下。

8．堆栈的操作指令

什么是堆栈，我们前面已经介绍过了，那么堆栈是如何进行数据传递的呢？对堆栈的操作指令有 2 条：

（1）PUSH　direct

（2）POP　direct

第 1 条指令称之为推入，就是将 direct 中的内容送入到堆栈中；第 2 条指令称之为弹出，就是将堆栈中的内容送回到 direct 中。这不难理解，我重点解释一下推入和弹出的执行过程：首先将 SP 中的值加 1，然后把 SP 中的值当作地址，将 direct 中的值送进以 SP 中的值为地址的 RAM 单元中。

```
例如：MOV   SP，#5FH ；
      MOV   A，#100  ；
      MOV   B，#20   ；
      PUSH  ACC      ；
      PUSH  B        ；
```

这段指令的执行过程是这样的：将 SP 中的值加 1，即变为 60H，然后将 A 中的值（#100）送到 60H 单元中，因此执行完 PUSH ACC 这条指令后， 内存 60H 单元的值就是 100，同样，执行 PUSH　B 时，是

将 SP+1，即变为 61H，然后将 B 中的值送入到 61H 单元中，即执行完本条指令后，61H 单元中的值变为 20。这是推入，那么弹出又是怎么样的呢？继续看下面的例子：

```
MOV   SP，#5FH   ；
MOV   A，#100    ；
MOV   B，#20     ；
PUSH   ACC       ；
PUSH   B         ；
POP   B          ；
POP   ACC        ；
```

POP 指令的执行是这样的：首先将 SP 中的值作为地址，并将此地址中的数送到 POP 指令后面的那个 direct 中，然后 SP 减 1。上面程序的执行过程是：将 SP 中的值（现在是 61H）作为地址，取 61H 单元中的数值（现在是 20），送到 B 中，所以执行完 POP　B 指令后 B 中的值是 20，然后将 SP 减 1，那么此时 SP 的值就变为 60H，然后执行 POP　ACC，将 SP 中的值（60H）作为地址，从该地址中取数（现在是 100），并送到 ACC 中，所以执行完本条指令后，ACC 中的值是 100。

这有什么意义呢？ACC 中的值本来就是 100，B 中的值本来就是 20 啊，是的，在本例中，的确没有意义，但在实际工作中，推入堆栈结束后（即执行指令 PUSH　B 后）往往要执行其他的指令，而且这些指令会把 A 中的值和 B 中的值改掉，所以在程序执行结束后，如果我们要把 A 和 B 中的值恢复到原来的值，那么这些指令就有意义了，具体应用我们将在以后的课程中讲到。这里还有一个问题，如果不用堆栈，比如说在 PUSH　ACC 指令处用 MOV　60H，A，在 PUSH　B 处用指令 MOV　61H，B，然后用 MOV　A，60H，MOV　B，61H 来替代两条 POP 指令，不也一样吗？是的，从结果上看是一样的，但是从过程看是不一样的，PUSH 和 POP 指令都是单字节，单周期指令，而 MOV 指令则是双字节，双周期指令。更何况，堆栈的作用不止于此，所以一般的单片机上都设有堆栈，而我们在编写子程序，需要保存数据时，通常也采用堆栈的方法来实现。

9．其他的数据传递类指令

（1）XCH　A，Rn

（2）XCH　A，direct

（3）XCH　A，@Ri

（4）XCHD　A,@Ri

（5）MOVC　A，A+PC

前面的 4 条指令是进行数据交换用的，第 1 条，寄存器与累加器交换；第 2 条直接地址与累加器交换；第 3 条间接 RAM 与累加器交换；第 4 条间接 RAM 与累加器的低 4 位交换；第 5 条是累加器与代码字节之间的数据传递类指令，这些指令作为初学者可能暂时还用不上，所以就不介绍了，大家只要了解一下就可以了，等下册中我们再来详细的讨论。

三．本课总结

到本课为止，数据传递类指令全部讲解完了，在单片机的指令中，数据传递类指令是使用最多的指令，因此这部分的内容是必须掌握的，如何来使用这些指令，我们将在以后的课程中结合具体的实验加以介绍。为了加深印象，大家可以用 DBG8051 软件对上述指令进行反复练习，用实验结果来加深课堂知识。

四．第 12 课习题

1．单片机是如何区分片外 RAM 和片外 ROM 的？又是如何区分片外 ROM 和片内 ROM 的？

2．简述推入堆栈和弹出堆栈的操作过程。

第十三课 单片机的指令（三）

算术、逻辑运算类指令也是单片机中极为重要的指令系统，在很多教科书中都把它们归为一类，实际上它们还是有区别的，为了让大家便于记忆，这里把它们分了开来。在 51 单片机中，算术运算类指令有 24 条；逻辑运算类指令有 25 条。这一课我们先来讲解算术运算类指令，下面我们分别加以讲解：

一．算术运算类指令

1。不带进位的加法指令

（1）ADD A，Rn ；例：ADD A，R7
（2）ADD A，@Ri ；例：ADD A，@R1
(3) ADD A，direct；例：ADD A，30H
（4）ADD A，#data ；例：ADD A，#30H

指令说明：这些指令的意思就是把后面的值与 A 中的值相加，结果送到 A 中去。

举例：MOV A，30H ；
ADD A，10H ；

执行结果 A=40H

2。带进位的加法指令

（1）ADDC A，Rn ；例：ADDC A，R7
（2）ADDC A，@Ri ；例：ADDC A，@R1
（3）ADDC A，direct；例：ADDC A，30H
（4）ADDC A，#data ；例：ADDC A，#30H

指令说明：这些指令的作用都是将 A 中的值和其后面的值相加，并且加上进位位 CY 中的值。为什么要这样做呢？我们知道 51 单片机是一种 8 位单片机，所以只能做 8 位的数学运算，也就是说最大运算的范围只能是 0-255，这在实际工作中肯定是不够的，因此就要进行扩展，怎么扩展，就是将 2 个 8 位的数学运算合起来，成为一个 16 位的运算，这样可以表达的数的范围就能达到 0-65535。如何合并呢？其实很简单，让我们看一个十进制数的加法例子：66+78，这两个数相加，我们根本不会在意它的过程，但事实上我们是这样做的：先做 6+8（低位），然后再做 6+7，这是高位。做了两次加法，只是我们做的时候并没有刻意分成两次加法来做罢了，或者说我们并没有意识到我们做了两次加法，之所以要分成两次来做，是因为这两个数超过了一位数所能表达的范围（0-9）。在做低位时产生了进位，我们通常的办法是在适当的位置点一下，然后在做高位加法时将这一点加进去；其实计算机中做 16 位加法时同样如此，先做低 8 位的，如果两数相加产生了进位，也要“点一下”做个标记，这个标记就是进位位 CY，在 PSW 中，我们前面已经讲过，在进行高位加法时将这个 CY 加进去。例如做 2 个 16 进制数相加：1067H+10A0H，先做 67H+A0H=107H，而 107H 显然超过了 0FFH，因此最终保存在 A 中的是 7，而 1 则进到了 PSW 中的 CY 位去了，换言之，CY 位就相当于是 100H，然后再做 10H+10H+CY，结果是 21H，所以最终的结果是 2107H。

3。带借位的减法指令

（1）SUBB A，Rn ；
（2）SUBB A，@Rn ；
（3）SUBB A，direct；
（4）SUBB A，#data ；

指令说明：没有不带借位的减法指令，如果需要做不带借位的减法指令（在做第一次相减时），只要将 CY 清零即可。

4．乘法指令

（1）MUL AB ；

指令说明：此指令的功能是将 A 和 B 中的两个 8 位无符号数相乘，两数相乘结果一般比较大，因此最终结果用 1 个 16 位数来表达，其中高 8 位放在 B 中，低 8 位放在 A 中。在乘积大于 FFFFFH（65535）时，PSW 的 0V 位置“1”（溢出），否则 OV 为“0”，而 CY 位总是为“0”。

例：（A）=4EH，（B）=5DH；

```
MUL   AB                    ;
```

乘积是 1C56H，所以在 B 中放的是 1CH，而 A 中放的则是 56H。

5．除法指令

（1）DIV AB（A/B）

指令说明：此指令的功能是将 A 中的 8 位无符号数除以 B 中的 8 位无符号数（什么是无符号数？简单的说就是没有负数的数，也就是整数，比如 1，2，3，1.2,4.5 等等这样的数）。除法一般会出现小数，但计算机中可没法直接表达小数，它用的是我们小学生用的商和余数的概念，如 13/5，其商是 2，余数是 3。除了以后，商放在 A 中，余数放在 B 中。CY 位和 OV 位都是“0”，如果在做除法前 B 中的值是 00H，也就是除数为 0，那么 0V=1。

6．加 1 指令

```
（1）INC   A      ；例如：A=20H      INC   A;       A=21H
（2）INC   Rn     ；例如：R7=20H     INC   A;       R7=21H
（3）INC   direct；例如：30H=20H    INC   30H;     30H=21H
（4）INC   @Ri    ；例如：
（5）INC   DPTR   ；例如：DPTR=20H   INC   DPTR;    DPTR=21H
```

指令说明：从结果上看 INC A 和 ADD A，#1 差不多，但 INC A 是单字节单周期指令，而 ADD A，#1 则是双字节双周期指令，而且 INC A 不会影响 PSW 位，如（A）=0FFH，INC A 后（A）=00H，而 CY 依然保持不变；如果是 ADD A，#1，则（A）=00H，而 CY 一定是“1”。因此加 1 指令并不适合做加法，事实上它主要是用来做计数、地址增加等用途。另外，加法类指令都是以 A 为核心的，其中一个数必须放在 A 中，而运算结果也必须放在 A 中，而加 1 类指令的对象则广泛得多，可以是寄存器、内存地址、间址寻址的地址等等。

7．减 1 指令

```
（1）DEC   A      ；例如：A=20H      DEC   A;       A=19H
（2）DEC   Rn     ；例如：R7=20H     DEC   A;       R7=19H
（3）DEC   direct；例如：30H=20H    DEC   30H;     30H=19H
（4）DEC   @Ri    ；例如：
```

指令说明：既然加 1 指令可以用于计数、定时、地址等加 1，那么有加也必然有减，所以减 1 指令的功能与加 1 指令类似，这里就不多说了。

8．十进制加法调整指令

```
DA   A   ；这是一条对十进制加法进行调整的指令，等下册用到时再介绍。
```

另外需要了解的是：在算术运算类指令中，除了加 1 和减 1 指令外，其他的算术运算类指令都要把结果放到累加器 A 中，这与数据传递类指令有所不同。

二．逻辑运算类指令

什么是逻辑运算？相信大家不会陌生，在数字电路中我们学过“与门”、“或门”、“非门”等，在单片机中也有类似的运算。那么它们是如何分类的呢？接下来我们就来一一讲解，先来看对累加器 A 的逻辑运算指令：

1。对累加器 A 的逻辑运算指令

（1）CLR A

指令说明：累加器 A 清零。效果同 MOV A，#00H 是一样的，只不过它是单周期指令，而 MOV A，#00H 是双周期指令。

（2）CPL A

指令说明：将累加器 A 逐位取反。相当于数字电路的“非”逻辑，例如：A=12H　CPL　A ；12H 化为二进制是 00010010，逻辑取反后为 11101101，即 A=EDH。

（3）RL　A

指令说明：将累加器 A 的值逻辑左移。例如：A=12H RL　A；化为二进制为 00010010，逐位左移后为 0010100，即 24H。这里把第 7 位移到了第 0，第 0 位移到了第 1 位，第 1 位移到了第 2 位，其余的依次类推。

（4）RLC　A

指令说明：加上进位位 CY 并逻辑左移。例如：CY=1 A=12H ；RLC　A；加上进位位 CY 后 1 00010010 逻辑左移变为 0 00100101（即 CY=0，A=25H）。

（5）RR　A

指令说明：将累加器 A 中的值逻辑右移。同 RL　A 类似。

（6）RRC　A

指令说明：加上进位位 CY 并逻辑右移。同 RLC　A 类似。

（7）SWAP　A

指令说明：将 A 中的值的高、低 4 位进行交换。例如：（A）=39H，SWAP　A 之后，A 中的值就是 93H。怎么正好是这么前后交换呢？因为这是一个十六进制数，每 1 个十六进位数代表 4 个二进制数。注意☺，如果是这样的：（A）=39D，后面没加 H，执行 SWAP　A 之后，可不是（A）=93。要将它化成二进制数再算：39D 化为二进制是 10111，也就是 0001，0111 高 4 位是 0001，低 4 位是 0111，交换后是 01110001，也就是 71H，即 113D。

2。两个寄存器之间的逻辑运算指令

上面的指令都是针对累加器 A 的逻辑运算指令，也就是说对一个寄存器的逻辑运算，那么如果两个寄存器之间的逻辑运算又是怎么样的呢？接着往下看：

```
（1）ANL   A,Rn          ；A 与 Rn 中的值按位‘与'，结果送入 A 中
     ANL   A,direct      ；A 与 direct 中的值按位‘与'，结果送入 A 中
     ANL   A,@Ri         ；A 与间址寻址单元@Ri 中的值按位‘与'，结果送入 A 中
     ANL   A,#data       ；A 与立即数 data 按位‘与'，结果送入 A 中
     ANL   direct,A      ；direct 中值与 A 中的值按位‘与'，结果送入 direct 中
     ANL   direct,#data ；direct 中的值与立即数 data 按位‘与'，结果送入 direct 中。
```

指令说明：什么是逻辑“与”？数字电路中我们已经学过：就是 F=A*B，简记为“全 1 出 1，有 0 出 0”。如果忘了，没关系，找本书再看一下，这里就不详细的阐述了。

例如：71H 和 56H 相“与”，将两数写成二进制形式：（71H） 01110001 和（56H） 00100110。逐位相“与”结果 就是 00100000 即 20H，从上面的例子可以看出，两个参与运算的值只要其中有一个位上是“0”，则这位的结果就是“0”，两个同是“1”，结果才是“1”，是不是符合逻辑“与”的结果？

知道了逻辑“与”指令的功能后，逻辑“或”和逻辑“异或”的功能就很简单了。逻辑“或”是逐位相“或”，即有 1 出 1，全 0 出 0。例：71H 和 56H 相“或”结果就是 77H；而“异或”则是逐位“异或”，即相同出 0，相异出 1。仍旧 71H 和 56H 相“异或”，结果是 57H。两个寄存器之间的逻辑“或”以及逻辑“异或”的指令如下：

```
（2）ORL   A,Rn       ；A 与 Rn 中的值按位‘或'，结果送入 A 中
     ORL   A,direct   ；A 与 direct 中的值按位‘或'，结果送入 A 中
     ORL   A,@Ri      ；A 与间址寻址单元@Ri 中的值按位‘或'，结果送入 A 中
     ORL   A,#data    ；A 与立即数 data 按位‘或'，结果送入 A 中
     ORL   direct,A   ；direct 中值与 A 中的值按位‘或'，结果送入 direct 中
     ORL   direct,#data ；direct 中的值与立即数 data 按位‘或'，结果送入 direct 中。
（3）XRL   A,Rn       ；A 与 Rn 中的值按位‘异或'，结果送入 A 中
     XRL   A,direct   ；A 与 direct 中的值按位‘异或'，结果送入 A 中
```

```
XRL   A,@Ri     ；A 与间址寻址单元@Ri 中的值按位‘异或'，结果送入 A 中
XRL   A,#data   ；A 与立即数 data 按位‘异或'，结果送入 A 中
XRL   direct,A  ；direct 中值与 A 中的值按位‘异或'，结果送入 direct 中
XRL   direct,#data；direct 中的值与立即数 data 按位‘异或'，结果送 direct 中。
```

连续好几节课将讲了许多的基本知识，大家是不是又觉得有些枯燥和无聊了，别急,接下来让我们轻松一下，做一个实验来证明一下几节课所学的内容：

三．LED 灯流动的实验，这是很经典的哦

1．实验程序：

```
 ORG   0000H          ;
LJMP   START          ;
ORG   30H             ;
START:MOV   SP,#5FH   ;
MOV   A,#80H          ;
LOOP:MOV   P1,A       ;
RL   A                ;
LCALL   DELAY         ;
LJMP   LOOP           ;
DELAY:MOV   R7,#255   ;
   D1:MOV   R6,#255   ;
   D2:NOP
      NOP
      NOP
      NOP
DJNZ   R6,D2          ;
DJNZ   R7,D1          ;
RET                   ;
END。
```

好久没做实验了，大家还记得实验的步骤吗？调试→编译→下载，看到了什么，有一个暗点在流动；想象一下，如果我们把 P1.0-1.7 的 LED 换成 8 只可控硅来控制霓虹灯，是不是就有点实用价值了。

2．程序分析：

前面的 ORG　0000H、LJMP　START、ORG　30H 我们都讲过了。从 START 开始，MOV　SP，#5FH，这叫初始化堆栈，在本程序中有无此句无关紧要，不过我们慢慢开始接触正规的编程，我也就慢慢地给大家培养习惯吧。MOV　A，#80H，将 80H 这个数送到 A 中去。干什么呢？不知道，往下看：MOV　P1，A 将 A 中的值送到 P1 端口去，此时 A 中的值是 80H，所以送出去的也就是 80H，因此 P1 口的值是 80H，也就是二进制 10000000，对应 P1.7-P1.0 这 8 位。我们应当知道，此时 P1.7 接的 LED8 是不亮的，而其它的 LED 都是亮的，所以就形成了一个“暗点”，继续往下看，RL　A；将 A 中的值进行左移，算一下，移之后的结果是什么？对了，是 01H，也就是二进制 00000001，这样，应当是接在 P1.0 上的 LED1 不亮了，而其它的都亮了，从现象上看就是“暗点”移到了后面；然后是调用延时程序，这里有一条指令 NOP，它是空操作指令，也就是什么都不做，用于短暂的延时，其他的指令我们很熟悉了，就是让这个“暗点”暗一会儿，然后又跳转到 LOOP 处（LJMP　LOOP），请大家计算一下，下面该哪个灯不亮了……对了，应当是接在 P1.1 上灯不亮了，这样依次不断的循环，就形成了“暗点流动”的现象。

3．提几个要求继续实验：

（1）如何实现亮点流动？

（2）如何改变流动的方向？

（3）如何实现几个灯的同时流动?

四．指令练习

结合前面所学习的知识自己进行练习。

请写出下列每条指令的执行结果，并用 DUG8051 软件进行验证，看结果是否正确。

```
MOV   A，#24H
MOV   R0，#37H
ORL   A，R0
XRL   A，#29H
MOV   35H，#10H
ORL   35H，#29H
MOV   R0，#35H
ANL   A，@R0
```

五．本课总结

本课的主要内容是算术运算类指令和逻辑运算类指令，在很多的教科书中都把它们归为一类，可见它们之间还是有一定联系的，大家可以自行找一下其中的规律。这里再罗嗦一句，学习单片机重在实践，希望大家多多动手哦，只有通过动手才能掌握课堂知识。

六．第 13 课习题

1．算术运算类指令和逻辑运算类指令的区别在哪里?

2．逻辑“与”，逻辑“或”，逻辑“异或”的运算结果是什么?

3．什么是有符号数? 无符号数?什么是整数?什么是浮点数?

4．计算一下 68H+ADH 的加法结果。

第十四课 单片机的指令（四）

一．控制转移类指令

控制转移类指令共有 17 条，分为无条件转移指令、条件转移指令和返回及调用指令三大类，下面我们分别加以学习：

1．无条件转移类指令

（1）无条件绝对转移指令：AJMP addr11

（2）无条件长转移指令：LJMP addr16

（3）无条件相对转移指令：SJMP rel

在讲解上面这三条指令之前先来认识一下三个符号：add11、add16、rel。其中 add11 和 add16 表示外部 ROM 的 16 位和 11 位地址，前面我们已经讲过，单片机的外部 ROM 可以扩展到 64K，add16 就表示 64K 程序存储器的任何地址，换句话说，LJMP 指令可以跳转到程序的任何地方，而 add11 则表示下一条指令的 2K 页面，也就是说，SJMP 指令只能跳转到程序的 2K 范围之内；rel 表示 8 位的偏移量，其范围是下一条指令第一字节的前 128 到后 127 个字节(即-128-+127)。介绍完了三个符号，再看上面的三条转移类指令，如果要仔细分析的话，它们之间其实区别很大，但在初学时，我们可以不理会这么多，统统把它们理解成：（*JMP 标号），比如 SJMP LOOP，就是跳转到有 LOOP 标号处。原则上，所有用 SJMP 或 AJMP 的地方都可以用 LJMP 来替代。因此在初学时，需要跳转时可以全不采用 LJMP 代替。下面再来看第 4 条跳转指令：

（4）无条件间接转移指令：JMP @A+DPTR

这条指令的用途也是跳转，跳转到什么地方去呢？这可不能由标号简单地决定了，让我们从一个实际的例子入手吧：

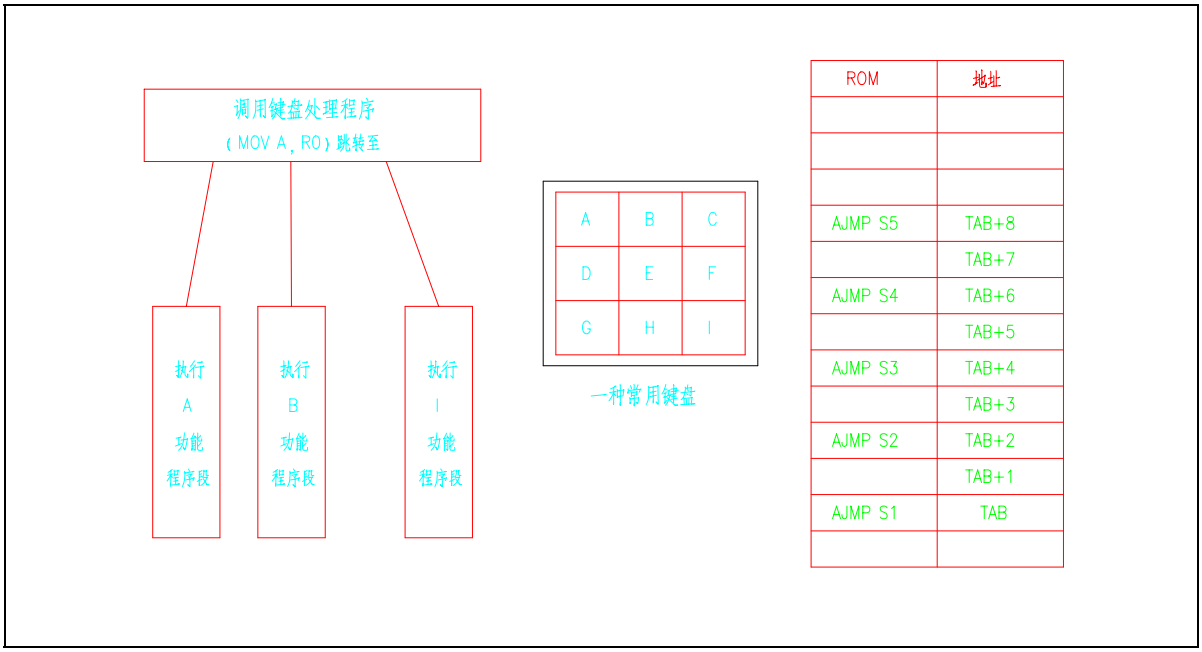
```
MOV   DPTR，#TAB ；将 TAB 所代表的地址送入 DPTR
MOV   A，R0      ；从 R0 中取数（详见下面说明）
MOV   B，#2      ；
MUL   A，B       ；
A 中的值乘 2     ；（详见下面的说明）
JMP   A，@A+DPTR ；跳转
TAB: AJMP   S1   ；跳转表格
AJMP  S2         ；
AJMP  S3         ；
。
。
。
。
。
```

应用背景介绍：在单片机开发中，经常要用到键盘，见下面的 9 个按键的键盘图。我们的要求是：当按下功能键 A……G 时去完成不同的功能，这用程序设计语言来表达的话，就是：按下不同的键去执行不同的程序段，以完成不同的功能，怎么样来实现这个功能呢？

看图，前面的程序读入的是按键的值，如按下‘A'键后获得的键值是“0”，按下‘B'键后获得的值是“1”等等，然后根据不同的值进行跳转，如键值为“0”就转到 S1 处执行，如键值为“1”就转到 S2 处执行，……到底如何来实现这一功能呢？

先从程序的下面看起，是若干条 AJMP 语句，这若干条 AJMP 语句最后在存储器中是这样存放的（见图），也就是每个 AJMP 语句都占用了两个存储器的空间，并且是连续存放的。而 AJMP S1 存放的地址是

TAB，到底 TAB 等于多少，我们不需要知道，把它留给汇编程序来算好了。



下面我们来看这段程序的执行过程：第 1 条 MOV　DPTR，#TAB 执行完了之后，DPTR 中的值就是 TAB，第 2 条是 MOV　A，R0，我们假设 R0 是由按键处理程序获得的键值，比如按下‘A’键，R0 中的值是“0”，按下‘B’键，R0 中的值是“1”……以此类推；现在我们假设按下的是‘B’键，则执行完第 2 条指令后，A 中的值就是“1”。并且按照我们的分析，按下‘B’后应当执行 S2 这段程序，让我们来看一看是否是这样呢？第 3 条、第 4 条指令是将 A 中的值乘“2”，即执行完第 4 条指令后 A 中的值是“2”，下面就执行 JMP　@A+DPTR 了，现在 DPTR 中的值是“TAB”，而 A+DPTR 后就是“TAB+2”，因此，执行完这条程序后，将会跳到 TAB+2 这个地址处继续执行；看一看在 TAB+2 这个地址里面放的是什么？就是 AJMP S2 这条指令，因此，马上又执行 AJMP　S2 这条指令，程序将跳到 S2 处往下执行，这与我们的要求相符合。请大家自行分析按下键‘A’、‘C’、‘D’……之后的情况。

这样我们用 JMP　@A+DPTR 这条指令就实现了按下一个键跳转到相应程序段去执行的这样一个要求。再提一个问题，为什么取得键值后要乘“2”呢？如果例程下面的所有指令换成 LJMP，即：LJMP S1,LJMP S2……这段程序还能正确地执行吗？如果不能，应该怎么改？

2．条件转移类指令

条件转移类指令就是在满足一定的条件后进行相对转移。

（1）累加器为 0 转移指令：JZ　rel

（2）累加器非 0 转移指令 ：JNZ　rel

第 1 条指令的功能是：如果(A)=0，则转移，否则顺序执行（执行本指令的下一条指令），转移到什么地方去呢？如果按照传统的方法，就要算偏移量，很麻烦，好在现在我们可以借助于机器汇编了。因此这条指令我们可以这样理解：JZ 标号，即转移到标号处。下面举一个例子来加以说明：

```
MOV　A，R0　　;
JZ　L1　　;
MOV　R1,#00H　　;
AJMP　L2　　;
L1: MOV　R1,#0FFH;
L2: SJMP　L2　　;
END
```

在执行上面这段程序前如果 R0 中的值是“0”的话，就转移到 L1 标号处执行，因此最终的执行结果是 R1 中的值为 0FFH；而如果 R0 中的值不等于“0”，则顺序执行，也就是执行 MOV　R1，#00H 指令，

最终的执行结果是 R1 中的值等于“0”。把这个例子中的 JZ 改成 JNZ 试试吧，看看程序执行的结果是什么？（这里注意一点：L1：MOV R1，#0FFH 这条指令的位置不能大于-128 到+127，为什么，我好象刚才讲过了。

（3）比较转移指令

A. CJNE A，#data,rel

B. CJNE A,direct,rel

C. CJNE Rn,#data,rel

D. CJNE @Ri,#data,rel

指令说明：第 1 条指令的功能是将 A 中的值和立即数 data 比较，如果两者相等，就顺序执行；如果不相等，就转移。同样地，我们可以将 rel 理解成标号，即：CJNE A，#data,标号。这样利用这条指令，我们就可以判断两数是否相等，这在很多场合是非常有用的，但有时还想知道两数比较之后哪个大，哪个小，本条指令也具有这样的功能，如果两数不相等，则 CPU 还会反映出哪个数大，哪个数小，这是用 CY（进位位）来实现的，如果前面的数（A 中的）大，则 CY=0；否则 CY=1，因此在程序转移后再次利用 CY 就可判断出 A 中的数比 data 大还是小了。

```
例如：MOV  A,R0        ;
      CJNE  A,#10H,L1  ;
      MOV  R1,#0FFH    ;
      AJMP  L3         ;
  L1: JC  L2           ;
      MOV  R1,#0AAH    ;
      AJMP  L3         ;
  L2: MOV  R1,#0FFH    ;
  L3: SJMP  L3         ;
```

上面的程序中有一条指令我们还没学过，即 JC，这条指令的原型是 JC rel,作用和上面的 JZ 类似，但是它是判 CY 是“0”还是“1”进行转移，如果 CY=1，则转移到 JC 后面的标号处执行；如果 CY=0 则顺序执行。

分析一下上面的程序，如果（A）=10H，则顺序执行，即 R1=0；如果（A）不等于 10H，则转到 L1 处继续执行，在 L1 处，再次进行判断，如果（A）>10H，则 CY=1，将顺序执行，（即执行 MOV R1，#0AAH 指令）；而如果（A）<10H，则将转移到 L2 处指行，（即执行 MOV R1，#0FFH 指令）。因此最终结果是：本程序执行前，如果（R0）=10H，则（R1）=00H；如果（R0）>10H，则（R1）=0AAH；如果（R0）<10H，则（R1）=0FFH。

弄懂了这条指令，其它的几条就类似了，第 2 条是把 A 当中的值和直接地址中的值比较，第 3 条则是将直接地址中的值和立即数比较，第 4 条是将间址寻址得到的数和立即数比较，这里就不解释了，请大家自行分析一下。下面给出几个相应的例子：

```
CJNE  A,10H;      把 A 中的值和 10H 中的值比较（注意和上题的区别）
CJNE  10H，#35H;  把 10H 中的值和 35H 中的值比较
CJNE  @R0,#35H;   把 R0 中的值作为地址，从此地址中取数并和 35H 比较
```

（4）循环转移指令

A. DJNZ Rn,rel

B. DJNZ direct,rel

第 1 条指令在前面的实验中已经有详细的分析，这里就不解释了；第 2 条指令，只是将 Rn 改成 direct，其它的也一样。

3．调用及返回指令

在前面的实验中，我们已用过了子程序，只是我们并没有明确地介绍子程序是干什么用的，为什么要用子程序呢？举个例子，我们数学老师布置了 10 道算术题，经过观察，每一道题中都包含一个（5+2）

*3 的运算。我们可以有两种选择，第一种选择，每做一道题，都把这个算式算一遍；第二种选择，我们可以先把这个结果算出来（也就是 21），放在一边，然后要用到这个算式时就将 21 代进去，这两种方法哪种更好呢？那就不必多言了吧。设计程序时也是这样，有时候一个功能会在程序的不同地方反复使用，我们就可以把这个功能设计成一段程序，每次需要用到这个功能时就“调用”一下，这就是子程序的用途所在，当然实际上子程序的作用还远不止这些，后面我们还会更详细地讲解子程序的用法。

主程序调用了子程序，子程序执行完之后必须再返回到主程序继续执行，不能“一去不回头”，那么回到什么地方呢？就是回到调用子程序的下面一条指令处继续执行。大家可以回去看一下前面的实验，是不是这样做的？

（1）调用指令

A. 长调用指令：LCALL addr16

B. 短调用指令：ACALL addr11

上面两条指令都是在主程序中调用子程序，两者的区别大家结合前面的知识可以自己分析一下。同样作为初学者，我们可以不必加以区分，也就是说可以用 LCALL 指令代替 ACALL。

（2）返回指令

A. 子程序返回：RET

B. 中断返回：RETI

RET 用于子程序返回，RETI 用于中断程序返回，这是有区别的，可不能用错了，至于什么是中断，我们讲到中断时再来解释。如何从子程序返回呢？很简单，就是执行 RET 指令，看一下前面的实验。

4. 空操作指令

所谓空操作，就是什么事也不干，停一个机器周期，一般用作短时间的延时，不要以为它没用，其实作用还是不小的，等你入门了就知道了。

二. 本课总结

本课的内容是控制转移类指令，这些指令在单片机中起着非常重要的作用，请大家务必搞清楚它们的作用和功能。

三. 第 14 课习题

1. 控制转移类指令分为几大类？AJMP，LJMP，SJMP 的区别是什么？
2. 什么是子程序？
3. 用实验仪编一段程序，要求按一个键盘时 LED1 闪一秒，按另一个按键后 LED1 闪 0.5 秒。

第十五课 单片机的指令（五）

一．位及位操作指令

位操作指令也叫布尔操作指令。什么是布尔指令？它有什么用呢？这个问题稍微有点复杂，我只能给大家简单的介绍一下。在 MCS-51 系列单片机中，有一个功能很强的布尔处理器，它实际上是一个独立的一位处理器，它有一套专门处理布尔变量（布尔变量也叫开关变量，就是以位作为单位的运算和操作）的指令子集，以完成对布尔变量的传送、运算、转移、控制等操作，这个子集的指令就是布尔操作指令。那么为什么要有这样的一套的指令系统？它是如何操作的呢？大家接着往下看：

1．位寻址的概念

为什么要位寻址呢？单片机不是可以有多种寻址方式吗？大家是否还记得，我们第十三课做的那个流水灯实验，用的就是“位”操作，也就是对一盏灯的亮和灭进行控制，而之前我们学的指令却全都是用“字节”来介绍的：字节的移动、加减法、逻辑运算、移位等等，用字节来处理一些数学问题（比如控制空调的温度、电视机的音量等等）非常直观，可以直接用数值来表示；可是如果用它来控制一个开关的打开或者合上，一个灯的亮或者灭，就有些不直接了。比如我们前面课上的那个流水灯的实验，我们把数值送往 P1 口之后并不能马上知道是哪个 LED 灭了，而是要化成二进制后才能知道。在工业控制中有很多场合需要处理这类单个的开关输出，比如一个继电器的吸合或者释放、一个指示灯的亮或者灭，用字节来处理就显得有些麻烦了，所以在 51 系列单片机中就特意引入了一个位处理机制。那么位处理器有多少地址空间？哪些特殊功能寄存器可以直接进行位寻址呢？

2．可位寻址的特殊功能寄存器

在 89C51 单片机中，内部 RAM 的范围是 00H-7FH 之间，其中可位寻址的低 128 位处于内部 RAM 的 20H-2FH 字节单元，其位地址从 00H-7FH，看下面的表：

字节地址	位地址							
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EF	77H	76H	75H	74H	73H	72H	71H	70H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

在物理实体上它们与原来的以字节寻址的 RAM 及端口是完全一样的，换句话说这些 RAM 单元及端口都可以有两种用法，比如用 28H.5 和 45H 是一样的。

除此之外，从 80H 单元开始除了程序计数器 PC 和 4 个工作寄存器区外，每 8 个字节还安排了 21 个特殊功能寄存器（89C52 有 26 个），这些 SFR 都有一个共同的特点：就是其字节地址均可被 8 整除，大家回到前面看一下第九课的表格。这些 SFR 都是具有位寻址功能的，也就是说这些 RAM 单元的每一个位都可以直接用这个地址来对其直接进行操作。了解了位操作的原理，再来看位操作的指令：

4．位操作指令

（1）位传送指令

A．MOV　C，bit

B．MOV　bit，C

指令说明：这两条指令的功能是实现进位位和其它位地址之间的数据传递（这里 bit 就是位的意思）。例如：MOV P1.0,CY；将 CY 中的状态送到 P1.0 引脚上去（如果是做算术运算，我们就可以通过观察 P1.0 端口知道现在 CY 是多少了）。再如：MOV　P1.0,CY；将 P1.0 的状态送给 CY。

（2）位清零指令

A．CLR　C

B．CLR　bit

指令说明：第 1 条指令使 CY=0；第 2 条指令使指定的位地址等于“0”。例如：CLR　P1.0，使 P1.0 为“0”。

（3）位置 1 指令

A．SETB　C

B．SETB　bit

指令说明：第 1 条使 CY=1；第 2 条使指定的位地址等于“1”，例如：SETB　P1.0，使 P1.0 为“1”。

（4）取反指令

A．CPL　C

B．CPL　bit

指令说明：第 1 条使 CY 等于原来的相反的值，即由“1”变为“0”，由“0”变为“1”；第 2 条使指定位的值等于原来相反的值，（相当于做“非”运算）。例如：CPL　P1.0，以我们做过的实验为例，如果原来灯是亮的，则执行本指令后灯就灭了；反之就是灯亮。

（5）位逻辑“与”指令

A．ANL　C,bit

B．ANL　C,/bit

指令说明：第 1 条 CY 位与指定的位地址的值相“与”，结果送回 CY；第 2 条先将指定的位地址中的值取出后取反，再和 CY 相“与”，结果送回 CY，但需注意☺，指定的位地址中的值本身并不发生变化。例如：ANL　C,/P1.0　设：执行本指令前，CY=1，P1.0 等于“1”（灯灭），则执行完本指令后 CY=0，而 P1.0 仍等于“1”。可用下列程序进行验证：

```
ORG　0000H　　;
AJMP　START　　;
ORG　30H　　;
START：MOV　SP，#5FH ;
　　MOV　P1，#0FFH;
　　SETB　C　　;
　　ANL　C，/P1.0 ;
　　MOV　P1.1,C　 ；将做完的结果送 P1.1,结果应当是 P1.1 上的灯亮，而 P1.0 上的灯还是亮。
```

（6）位逻辑“或”指令

A．ORL　C，bit

B．ORL　bit，C

这两条指令的功能大家自行分析吧，然后对照上面的例程，自己编一个验证程序，看看自己想得对不对？

（7）判 CY 条件转移指令

A．JC　rel

B. JNC rel

指令说明：这两条指令叫做判 CY 转移指令，第 1 条指令的功能是如果 CY 等于“1”就转移；如果不等于“1”就顺序执行，那么转移到什么地方去呢？我们可以这样理解：JC 标号，即如果等于“1”就转到标号处执行；第 2 条指令则和第 1 条指令正好相反，即如果 CY=0 就转移，不等于“0”则顺序执行，转移到什么地方，我们同样可以这样理解：JNC 标号。

（8）判位变量转移指令

A. JB bit,rel

B. JNB bit,rel

指令说明：第 1 条指令是如果指定的 bit 位中的值是“1”，则转移；否则就顺序执行，转移到什么地方，同样我们可以这样理解：JB bit,标号；第 2 条指令请大家自行分析一下。

（9）判位变量转移并将该位清零：JBC bit，rel

指令说明：这条指令同 JB bit，rel 的区别在于判“1”转移的同时清除该位，为什么要这样做呢？以后我们会讲到。

接下来我们做一个这方面的实验：

二. 位操作指令实验

1. 实验程序：

```
ORG   0000H            ;
LJMP   START           ;
ORG   30H              ;
START:MOV   SP,#5FH    ;
MOV   P1,#0FFH         ;
MOV   P3,#0FFH         ;
L1:JNB   P3.4,L2       ;P3.4 上接有一只按键，它按下时，P3.4=0
    JNB   P3.5,L3      ;P3.5 上接有一只按键，它按下时，P3.5=0
    LJMP   L1          ;
L2:MOV   P1,#00H       ;
    LJMP   L1          ;
L3:MOV   P1,#0FFH      ;
    LJMP   L1          ;
END.
```

2. 程序分析：

把上面的程序下载到实验板上，看看有什么现象……按下接在 P3.4 上的按键，P1 口的灯全亮了，松开或再按，灯并不熄灭；然后按下接在 P3.5 上的按键，灯就全灭了，这像什么？这不就是工业控制中经常用到的启动、停止功能吗？怎么做到的呢？一开始，将 0FFH 送入 P3 口，这样，P3 口所有的引线都处于高电平，然后执行 L1，如果 P3.4 是高电平（键没有按下），则顺序执行 JNB P3.5,L3 语句；同样，如果 P3.5 是高电平（键没有按下），则顺序执行 LJMP L1 语句，这样就不停地检测 P3.4 和 P3.5。如果有一次 P3.4 上的按键按下去了，则转移到 L2（执行 MOV P1，#00H），使灯全亮，然后又转去 L1，再次循环，直到检测到 P3.5 为“0”，就转去 L3（执行 MOV P1，#0FFH），使灯全灭，再转去 L1，如此不断地循环就可以了。这里提一个问题，我们这个实验中控制的是一个字节（既整个 P1 口），如何来实现一位（比如 P1.0）的控制呢？其实很简单，只要把程序改一下就可以了。

程序如下：

```
ORG   0000H            ;
LJMP    START          ;
ORG    30H             ;
START:MOV    SP,#5FH   ;
```

```
    MOV  P1,#0FFH       ;
    MOV  P3,#0FFH       ;
    L1:JNB  P3.4,L2     ;P3.4 上接有一只按键，它按下时，P3.4=0
       JNB  P3.5,L3     ;P3.5 上接有一只按键，它按下时，P3.5=0
       LJMP  L1         ;
    L2:CLR  P1.0        ;亮 LED1
       LJMP  L1         ;
    L3:SETB  P1.0       ;暗 LED1
       LJMP  L1         ;
    END。
```

尽管实际的程序还要考虑按键的去抖动问题，但程序的基本结构和流程是实用的，这样的程序就能完成我们对工业控制中一个继电器的控制目的。怎么样，如果现在让您用单片机控制一台电机的正反转应该没有问题了吧，试试看。最后提个问题：能不能把本程序改用 JB 指令来写，如果行的话，该怎么写呢？

既然讲到了位操作指令，就一定要讲讲位操作指令的写法，在很多的程序中，常常这样写程序：CLR 20H.02；CLR PSW.7；什么意思？以 PSW.7 为例，大家还记得 PSW 是什么？对了是程序状态字寄存器，那么 PSW.7 是什么意思呢？其实说穿了很简单就是把 PSW 的第 7 位清零，第 7 位是什么？忘了，也太没记性了，是 CY 位，回去翻翻！这样一来 CLR 20H.02 也应该懂了吧。

三．本课总结

这一课我们讲述了位操作指令的编程方法，事实上正是由于单片机有了位操作机制，才使得其编程变得十分的简单和方便，希望大家把这部分的内容搞清楚。

到本课为止，单片机的指令已经全部讲完，也许有些您已经记住了，而有些可能还记不住，或者说还不会使用。没关系，我们以后还会在具体的实验中再来讲解它们，下一课我将向大家讲解单片机程序的设计方法，这可是蛮重要的哦！

四．第 15 课习题

1．位寻址与字节寻址的区别在哪里？

2．单片机中哪些 RAM 和 SFR 是可以位寻址的？

3．写一段程序，使两个 LED 灯交替地亮或灭，类似于工业控制中的正反转功能。

第十六课 单片机程序设计方法

程序设计是单片机开发最重要的工作，程序设计就是利用单片机的指令系统，根据应用系统（即目标产品）的要求编写单片机的应用程序，其实我们前面已经开始这样做过了，这一课我们不是讲如何来设计具体的程序，而是教您设计单片机程序的基本方法。不过在讲解之前还是有必要先了解一下单片机的程序设计语言。

一．程序设计语言

这里的语言与我们通常理解的语言是有区别的，它指的是为开发单片机而设计的程序语言，如果您没有学过程序设计可能不太明白，我给大家简单解释一下，您知道微软的 VB，VC 吗？VB，VC 就是为某些工程应用而设计的计算机程序语言，通俗地讲，它是一种设计工具，只不过这种工具是用来设计计算机程序的。要想设计单片机的程序当然也要有这样一种工具（说设计语言更确切些），单片机的设计语言基本上有三类：

1．完全面向机器的机器语言

机器语言就是能被单片机直接识别和执行的语言，计算机能识别什么？以前我们讲过--是数字“0”或“1”，所以机器语言就是用一连串的“0”或“1”来表示的数字。比如：MOV A，40H；用机器语言来表示就是 11100101 0100000，很显然，用机器语言来编写单片机的程序不太方便，也不好记忆，我们必须想办法用更好的语言来编写单片机的程序，于是就有了专门为单片机开发而设计的语言：

2．汇编语言

汇编语言也叫符号化语言，它使用助记符来代替二进制的“0”和“1”，比如：刚才的 MOV A，40H 就是汇编语言指令，显然用汇编语言写成的程序比机器语言好学也好记，所以单片机的指令普遍采用汇编指令来编写，用汇编语言写成的程序我们就叫它源程序或源代码。可是计算机不能识别和执行用汇编语言写成的程序啊？怎么办？当然有办法，我们可以通过“翻译”把源代码译成机器语言，这个过程就叫做汇编，汇编工作现在都是由计算机借助汇编程序自动完成的，不过在很早以前，它是靠手工来做的，道听途说，我也没经历过，呵呵。

值得注意的是，汇编语言也是面向机器的，它仍是一种低级语言。每一类计算机都有它自己的汇编语言，比如：51 系列有它的汇编语言，PIC 系列也有它的汇编语言，微机也有它自己的汇编语言，它们的指令系统是各不相同的，也就是说，不同的单片机有不同的指令系统，它们之间是不通用的，这就是为什么世界上有很多单片机类型的缘故了。为了解决这个问题，人们想了很多的办法，设计了许多的高级计算机语言，而现在最适合单片机编程的要数 C 语言。

3．C 语言—高级单片机语言

C 语言是一种通用的计算机程序设计语言，它既可以用来编写通用计算机的系统程序，也可以用来编写一般的应用程序，由于它具有直接操作计算机硬件的功能，所以非常适合用来编写单片机的程序，与其他的计算机高级程序设计语言相比，它具有以下的特点：

（1）。语言规模小，使用简单

在现有的计算机设计程序中，C 语言的规模是最小的，ANSIC 标准的 C 语言一共只有 32 个关键字，9 种控制语句，然而它的书写形式却比较灵活，表达方式简洁，使用简单的方法就可以构造出相当复杂的数据类型和程序结构。

（2）。可以直接操作计算机硬件

C 语言能够直接访问单片机的物理空间地址（KEIL C51 软件中的 C51 编译器更具有直接操作 51 单片机内部存储器和 I/O 口的能力），亦可直接访问片内或片外存储器，还可以进行各种位操作。

（3）。表达能力强，表达方式灵活

C 语言有丰富的数据结构类型，可以采用整型、实型、字符型、数组类型、指针类型、结构类型、联合类型、枚举类型等多种数据类型来实现各种复杂数据结构的运算。利用 C 语言提供的多种运算符，我们可以组成各种表达式，还可以采用多种方法来获得表达式的值，从而使程序设计具有更大的灵活性。

（4）。可进行结构化设计

结构化程序是单片机程序设计的组成部分，C 语言中的函数相当于汇编语言中的子程序，KEIL C51 的编译器提供了一个函数库，其中包含有许多标准函数，如各种数学函数、标准输入输出函数等，此外还可以根据用户需要编制满足某种特殊需要的自定义函数。C 语言程序就是由许多个函数组成的，一个函数即相当于一个程序模块，所以 C 语言可以很容易地进行结构化程序设计。

（5）。可移植性

前面我们讲过，由于单片机的结构不同，所以不同类型的单片机就要用不同的汇编语言来编写程序，而 C 语言则不同，它是通过汇编来得到可执行代码的，所以不同的机器上有 80%的代码是公用的，一般只要对程序稍加修改，甚至不加修改就可以方便地把代码移植到另一种单片机中。这对于已经掌握了一种单片机的编程原理，又想用另一种单片机的人来说，可以大大地缩短学习周期，我们将在教程的下册中专门来讲解 C 语言的应用及其编程原理。

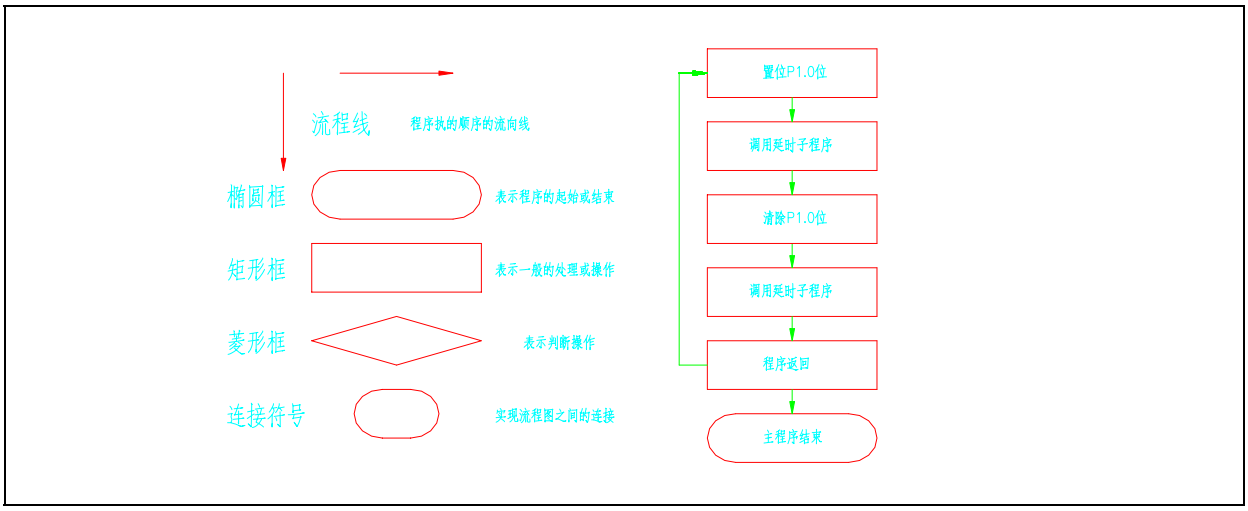
不过作为单片机初学者想要学会 C 语言也并不是一件容易的事，因此对于大多数人来说，汇编语言仍是编写单片机程序的主要语言。我们上册的教程将全部以汇编语言来编写单片机的程序。了解了单片机编程的设计语言，下面我们来看单片机编程的基本过程和步骤。

二．单片机程序设计的步骤

单片机的程序设计通常包括根据任务绘制程序流程图、编写程序及汇编等几个步骤。

1．绘制流程图

所谓流程图，就是用各种符号、图形、箭头把程序的流向及过程用图形表示出来。绘制流程图是单片机程序编写前最重要的工作，通常我们的程序就是根据流程图的指向采用适当的指令来编写的，下面的图形和箭头就是我们绘制流程图用的工具（图中左边所示）。



绘制流程图时，首先画出简单的功能流程图（粗框图），再对功能流程图进行扩充和具体化，即对存储器、标志位等单元做具体的分配和说明，把功能图上的每一个粗框图转化为具体的存储器或地址单元，从而绘制出详细的程序流程图，即细框图。下面举个例子给大家演示一下，请看下面的程序：

主程序：

```
LOOP:SETB   P1.0      ;
  LCALL   DELAY       ;
  CLR   P1.0          ;
  LCALL   DELAY       ;
  LJMP   LOOP         ;
```

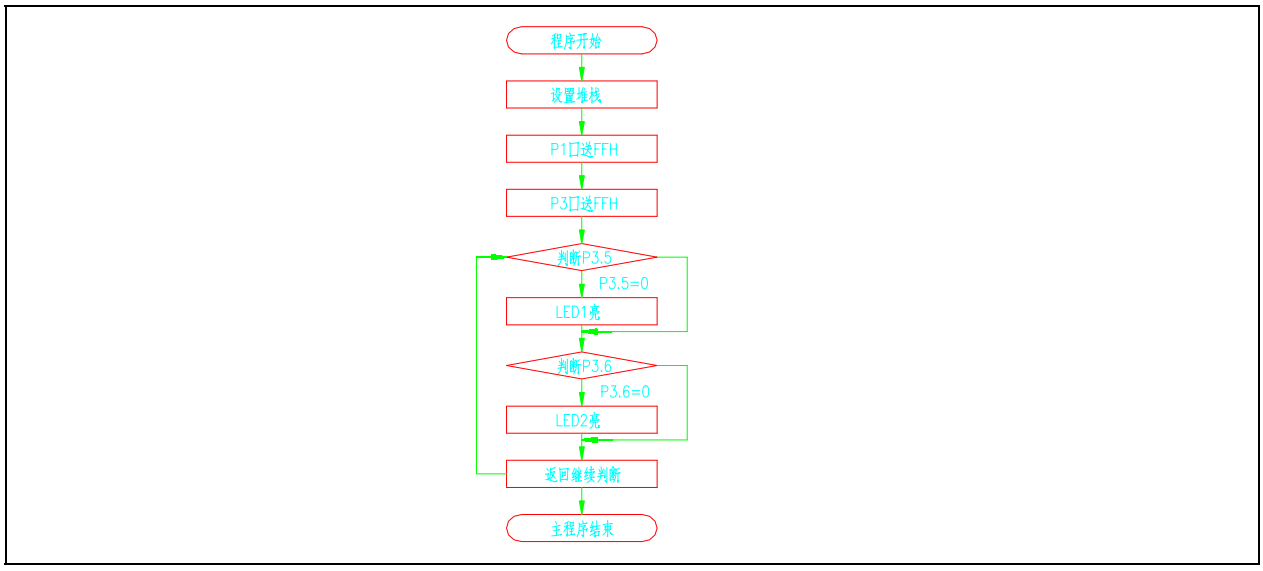
子程序：

```
DELAY: MOV   R7, #250;
   D1: MOV   R6, #250;
```

```
    D2：DJNZ  R6，D2 ；
    DJNZ  R7，D1     ；
    RET              ；
    END。
```

还记得吗，这是我们第四课中做过的 LED 灯闪烁的实验，以前我们曾对程序进行过分析，现在让我们用流程图来把这段程序的主程序部分画出来，看上图的右边部分。这就是程序的流程图，在单片机的编程过程中，绘制流程图能看清楚程序执行的步骤以及程序的流向，事实上，程序的编写就是根据流程图的功能完成的。下面我们来把第十五课中的那个程序也用流程图画出来。程序如下：

```
ORG  0000H          ；
LJMP   START        ；
ORG   30H           ；
START：MOV   SP，#5FH；
MOV  P1，#0FFH      ；
MOV  P3，#0FFH      ；
L1：JNB  P3.5，L2   ；P3.5 上接有一只按键，它按下时，P3.5=0
    JNB  P3.6，L3   ；P3.6 上接有一只按键，它按下时，P3.6=0
    LJMP  L1        ；
L2：CLR  P1.0       ；亮 LED1
    LJMP  L1        ；
L3：SETB  P1.0      ；暗 LED1
    LJMP  L1        ；
END。
```



先不看图，自己画一下，看是不是同我画的一样。在实际的程序设计中，根据框图，采用适当的指令编写出实现流程图的源程序就是我们编写程序的最后工作。

2．编写程序和汇编

程序编写完之后，我们要把它汇编成机器语言，这种机器语言就是十六进制文件，后缀名为*.HEX 文件，以前还要把它转换成二进制文件，后缀名为*.BIN 文件，不过现在的编程器都能直接读入十六进制文件，就不需要转换了，最后用编程器把程序写入单片机。这些以前都讲过了，这里就不重复了。下面来讲本课的主题—程序设计的方法。

三．单片机程序设计的方法

要想搞清楚程序设计的方法，我们首先要知道单片机到底有哪几类程序？单片机的程序分为结构

化程序、子程序和综合程序三个大类，先来看结构化程序。

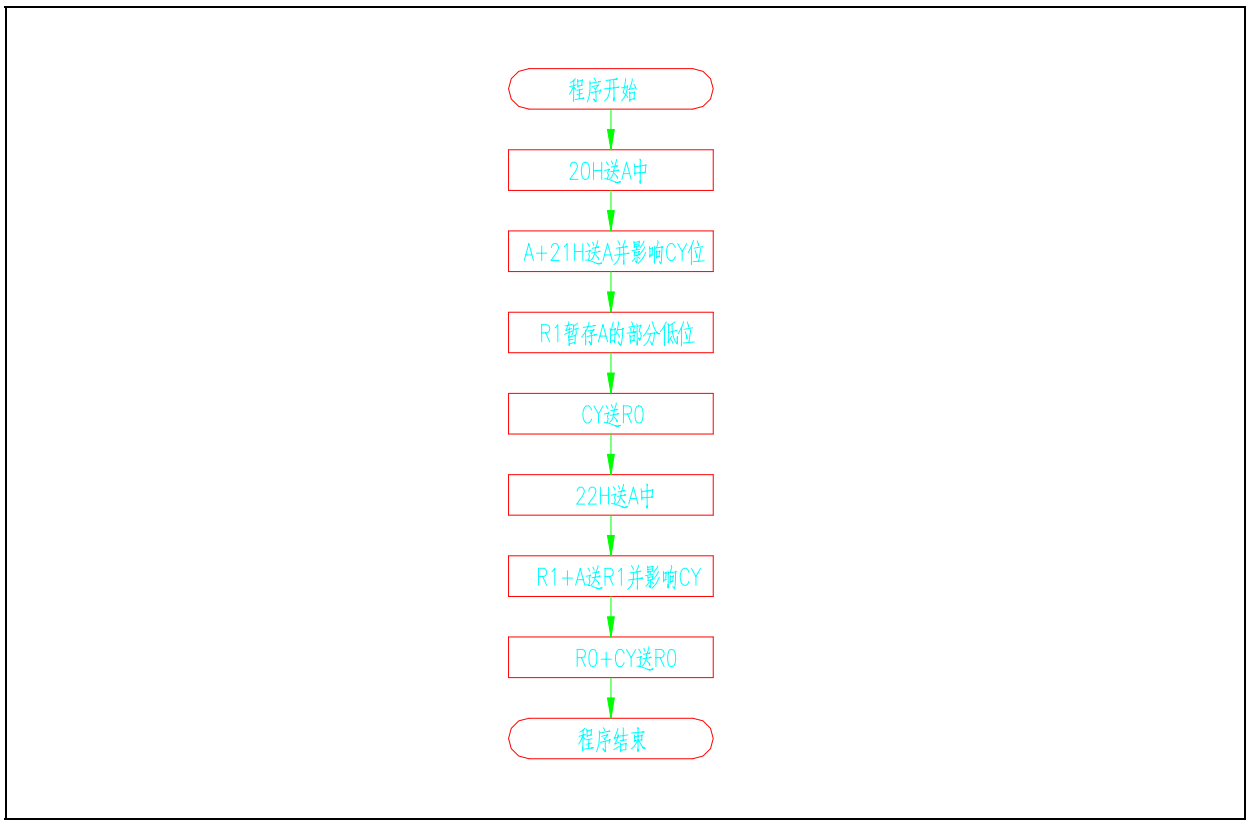
1．结构化程序的设计方法

在单片机的程序中，既有复杂的程序，也有简单的程序，但不论哪种程序，它们都是由一个个基本的程序结构组成的，这些基本结构有顺序结构、分支结构和循环结构。

（1）。顺序结构程序的设计

顺序结构的程序一般用来处理比较简单的算术或逻辑问题，它的执行过程是按照程序存储器 PC 自动加 1 的顺序执行的，主要用数据传递类指令和数据运算类指令来实现。比如我们前面第六课中的 I/O 口输入实验就是典型的顺序结构的程序。试试看，把这个程序的流程图写出来。下面再看一个例子：将内部 RAM 中 20H 单元和 30H 单元的无符号数相加，存入 R0（高位）和 R1（低位）中。

先画出流程图：



根据流程图编写源代码如下：

```
MOV   A，20H     ;
ADD   A，30H     ;
MOV   R0，A      ;
CLR   A          ;
ADDC   A，#00H   ;
MOV   R0，A      ;
MOV   A，30H     ;
ADD   A，R1      ;
MOV   R1，A      ;
CLR   A          ;
ADDC   A，R0     ;
MOV   R0，A      ;
```

这就是顺序结构程序，程序的原理就不分析了，我们接着讲分支结构的程序设计。

这里说明一点，最近有朋友提出这一课的有些程序看不懂，的确如此，这一课的有几个程序实例

我们从来没有学过，之所以放在这里，原本是为了让大家理解程序设计的方法，举几个示例证明一下，没想到反而增加了大家的难度。其实这些示例你不需要刻意的去理解它，只要明白它的设计方法就可以了，因为这一课的主要内容是程序设计的方法，而不是程序执行的原理和结果。如果以后有更好的示例我会修改一下。

（2）。分支结构程序的设计

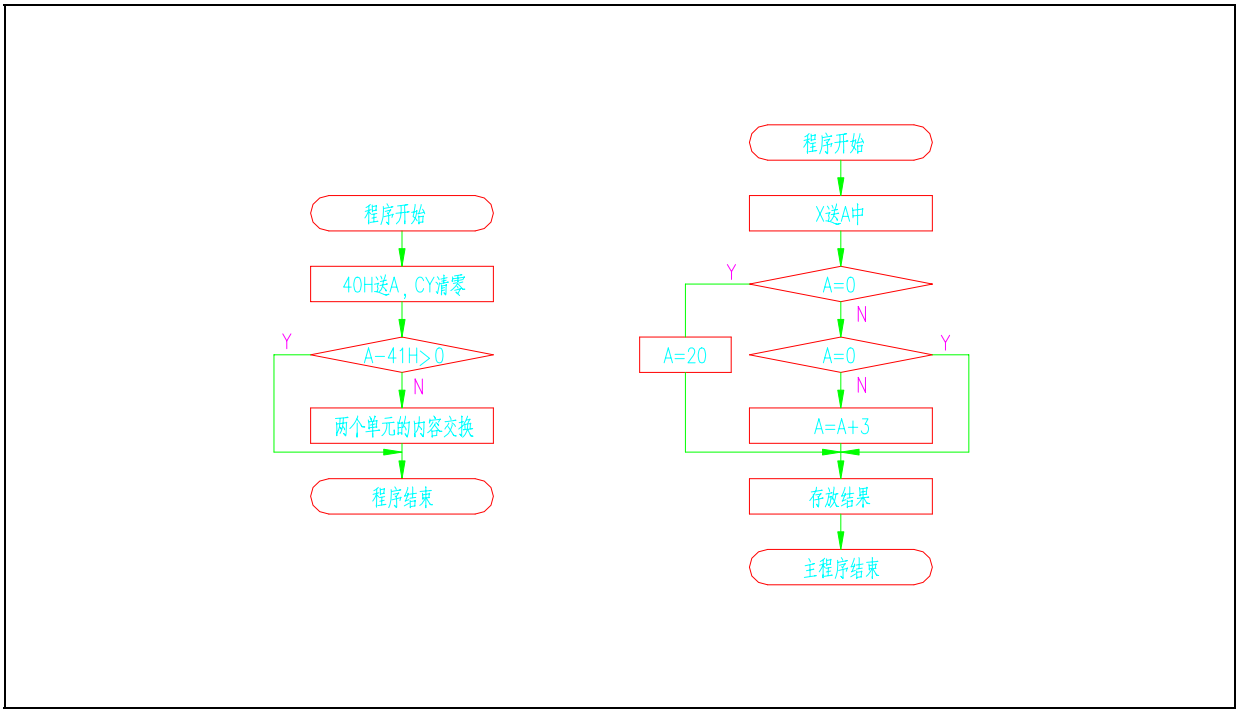
所谓分支结构就是利用条件转移指令，使程序执行某一指令后，根据所给的条件是否满足来改变程序执行的顺序，也就是本条指令执行完后，并不是象顺序结构那样执行下一条指令，而是看本条指令所给的条件是否满足，如果满足条件就跳转到其他的指令，如果不满足就顺序执行；当然也可以是满足条件顺序执行，而不满足条件跳转执行，看十五课实验程序中的下面两条：

```
L1：JNB   P3.5，L2     ；P3.5 上接有一只按键，它按下时，P3.5=0
    JNB   P3.6，L3     ；P3.6 上接有一只按键，它按下时，P3.6=0
```

这就是分支结构的程序，如果 P3.5 为“0”，就转移；反之就顺序执行。当然也可以改成 P3.5=0 顺序执行；而 P3.5=1 则转移，不过此时的程序就要用 JB 指令了。在 51 系列单片机中，可以直接用于分支程序的指令有 JB（JNB）、JC（JNC）、JZ（JNZ）、CJNE、JBC 等这几条，它们可以完成诸如正负判断、大小判断和溢出判断等等。在分支结构的指令设计中，大家必须注意☺：执行一条判断指令只可以形成两路分支，如果要形成多路分支，就必须进行多次判断，也就是多条指令连续判断。下面给大家举两个例子：

A．单分支结构的程序实例

假设有两个数在内部 RAM 单元的 40H 和 41H 中，现在要求找出其中较大的一个数，并将较大的数存入 40H 中，而将较小的一个数存入 41H 中。根据程序的要求，我们先画出程序的流程图（左图）。



再根据流程图写出程序的源代码如下：

```
MOV   A，40H    ；
CLR   C         ；
SUBB   A，41H   ；
JNC   WAIT      ；
MOV   A，41H    ；
XCH   A，41H    ；
```

```
MOV  40H，A   ；
WAIT：SJMP  WAIT；
END。
```

程序的原理请大家自行分析一下，接下来再举一个多分支结构的实例，看下面的程序：

```
MOV  A，20H          ；取数
JZ  ZERO             ；A=0，转移；A=1，顺序执行
JB  ACC。7，STORE    ；A 为负数，转移
ADD  A，#3           ；A 为正数，则加 3
SJMP  STORE          ；
ZERO：MOV  A，#20    ；
STORE：MOV  21H，A   ；
```

自己画一下本例的流程图，再和上面的右图比较一下，看是不是一样。这里有一条指令给大家解释一下：JB ACC.3，STORE；ACC.3 表示累加器 A 中的 D3 位，这条指令的意思就是看一下累加器中的 D3 位是正还是负，D3 是什么呢？在这里就是“0”（20H 的二进制 10000000）。上一课刚讲过，不要说又忘了？接下来再讲第三种循环结构的程序设计。

（3）。循环结构程序的设计

循环程序是最常用的程序结构形式，在单片机的程序设计中，有时要碰到一段程序需要重复执行多次的情况，此时就要用到循环结构程序，比如第四课中的实验--LED 灯闪烁程序的子程序：

```
DELAY：MOV  R7，#250；（1）
   D1：MOV  R6，#250；（2）
   D2：DJNZ  R6，D2 ；（3）
   DJNZ  R7，D1     ；（4）
   RET              ；（5）
   END。
```

在这段程序中，为了延时需要多次执行 DJNZ 指令，此时若用循环结构程序就可以大大地简化程序的设计，减少程序占用的存储器空间。循环结构指令一般有以下四个部分组成：

A．初始化部分

初始化部分主要用来设置循环的初始值，包括预值数、计数器和数据指针的初值。比如上例中的 #250 就是预值数初值。

B．循环处理部分

循环处理部分是程序的主体部分，也称为程序体，通过它可以完成程序处理的任务。

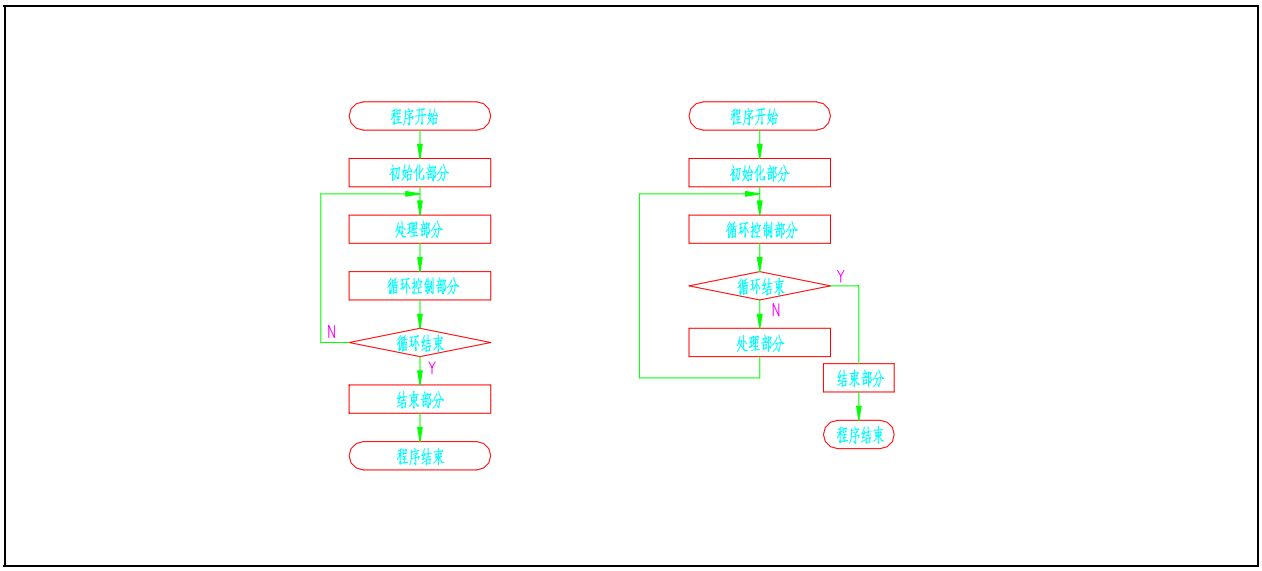
C．循环控制部分

循环控制部分可以控制程序循环的次数，并修改预值数或计数器和指针的值，检查该循环是否执行了足够的次数，如果到了足够的次数，就采用条件转移指令或判断指令来控制循环的结束。比如上例中的（3）、（4）指令就是当 R6 或 R7 中的值为“0”时就转移。

C．循环结束部分

循环结束后必须返回，一般用 RET 或 RETI 指令。这里注意☺：以上四个部分中，第一和第四部分只能执行一次，而第二和第三部分可以执行多次。

典型的循环结构程序的流程图可画成如下左图所示，也可以将处理部分和控制部分位置对调，如右图。在循环程序设计中，循环控制部分是程序设计的关键环节，常用的循环控制方式有计数器控制和条件控制两种。计数器控制就是把要循环的次数（即预值数）放入计数器中，程序每循环一次，计数器的值就减 1，一直到计数器的内容为零时，循环结束，一般用 DJNZ 指令；而条件控制方式常预先不知道要循环的次数，只知道循环的有关条件，此时就可以根据给定的条件标志位来判断程序是否继续，一般参照分支结构方法中的条件来判别指令并执行。下面举几个例子来分别解释一下，希望大家能以此类推。



程序一：用计数器控制的单重循环程序

源程序如下：

```
CLR  A              ;
MOV  R2，20H        ;
MOV  R1,；22H       ;
LOOP：ADD  A，@R1   ;
  INC  R1           ;
  DJNZ  R2，LOOP    ;
MOV  21H，A         ;
```

这段程序的作用是从 22H 单元开始存放一个数据块，其长度存放在 20H 单元中，将数据块求和，要求将和存放入 21H 单元中，和不超过 255。下面再举一个条件控制的循环程序。

程序二：用条件控制的单重循环程序

设字符串存放在内部 RAM 的 21H 开始的单元中，以结束作标志，要求计算出该字符串的长度，并将其存放在 20H 单元中。

源程序如下：

```
CLR  A           ;
MOV  R0，  #21H  ；将地址指针指向 21H 单元
LOOP：CJNZ  @R0，#24H，NEXT ；与  比较
  SJMP  COMP     ；找到结束
NEXT：INC  A     ；不为“0”，计数器加 1
  INC  R0        ；修改地址指针
  SJMP  LOOP     ;
COMP：MOV  20H，A  ；存放结果
```

试试看，自己把上面两段程序的流程图画出来。下面再看一个例子：

```
DELAY：MOV  R7，#250 ;
   D1：MOV  R6，#250 ;
   D2：DJNZ  R6，D2  ;
   DJNZ  R7，D1      ;
   RET               ;
   END。
```

这是一段约 125mS 的延时程序，现在我们来把它改成下面表格中的程序（右边的程序）：

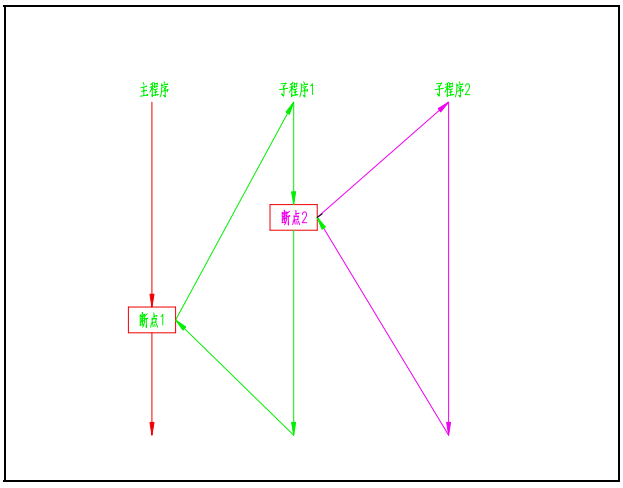
DELAY：MOV　R7，#250；	DELAY：MOV　R7，#250；
D1：MOV　R6，#250；	D1：MOV　R6，#250；
D2：DJNZ　R6，D2；	D2：MOV　R5，#250；
DJNZ　R7，D1；	D3：DJNZ　R5，D3 ；
RET；	DJNZ　R6，D2 ；
END。	DJNZ　R7，D1 ；
	RET；
	END。

从这里可以引出一个概念：程序的嵌套。什么是嵌套，比如早上我骑自行车从家里到单位去上班，当走到半路上时，太太叫我去孩子学校拿点东西；到了学校，老师又叫我把学校的一台电脑修一下；修好电脑，一个朋友又打电话叫我去他那里拿了一本《单片机与嵌入式系统》杂志，完了之后再去上班；这就是生活中的嵌套。在单片机的程序设计中，也有类似的现象，有时为了达到某个目的，往往要在一段循环程序中再加入另一段循环程序，这就是单片机的程序嵌套。通常我们把一个循环体中不再包含循环的叫做单重嵌套；如果一个循环体中还包括有循环，则叫做多重嵌套。上面的表格中左边的程序就是单重嵌套，而右边的程序则是多重嵌套。另外须注意☺：在多重嵌套中，不允许各个循环体互相交叉，也不允许从外循环跳入内循环，否则编译时会出错。了解了结构化程序的设计，下面再来看子程序的设计方法。

2．子程序的设计方法

什么是子程序？如何设计子程序？要解释这个问题，让我们先同样从生活中的一个例子说起，请看下面的数学题目：28*（33+65）+47*（33+65）+875*（33+65）。在这道题中，我们一般是怎么算的？也许大家都知道，一般总是先把（33+65）=98 代出来，然后再用（28+47+875）*98 来计算最后的结果，为什么会这样？这是因为在这道题中，我们多次用到了（33+65）这个中间结果。在单片机的程序设计中，有时也有这样的情况，比如下面的程序：

```
主程序
LOOP:SETB  P1.0        ；（1）
       LCALL  DELAY    ；（2）
       CLR  P1.0       ；（3）
       LCALL  DELAY    ；（4）
       LJMP  LOOP      ；（5）
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）
```



这是大家非常熟悉的 LED 灯延时程序，在这段程序中，两次调用到了 DELAY 这段程序，为了简化程序的设计，我们就把 DELAY 这段程序单独地列了出来，这段列出的程序我们就叫它子程序，而调用子程序的程序我们则叫它主程序（LOOP 的程序段）。在主程序执行时，每当要用到子程序时，我们就用 LCALL 指令来调用子程序，子程序执行完之后，必须返回主程序，返回就用 RET 指令，这我们以前都讲过了，这里不再重复。

另外，如果子程序执行的过程中，还要再次调用其他的子程序，这种现象我们就称它为子程序的嵌套。看上面右边的图，就是一个两层子程序的嵌套结构图。

这里有个问题？在子程序的执行过程中，有时可能要使用到累加器和某些工作寄存器，而在调用子程序前，这些寄存器中可能已经存放有主程序的中间结果，它们在子程序返回后仍要使用，这样就需

要在进入子程序之前，将要使用的累加器和寄存器中的内容预先转移到安全的地方保存起来，这叫现场保护；当子程序执行完即将返回主程序之前，还要将这些内容先取出来，送回到累加器和原来的工作寄存器中，这个过程叫恢复现场。

保护现场和恢复现场通常使用堆栈，即在进入子程序之前，将需要保护的数据压入堆栈，在返回之前再将压入的数据弹出到原来的工作单元中，恢复原来的状态。看下面的例子：

```
LOOP：PUSH   03H          ；将 03H 单元中的值压入堆栈保护
   PUSH   ACC             ；将累加器中的值压入堆栈保护
   ……
   ……
   POP    ACC             ；将 ACC 中的值从堆栈弹出
   POP    03H             ；恢复 03H 单元中的内容
   RET                    ；从子程序返回
```

由于堆栈的操作是“后进先出，先进后出”，所以编写指令时，必须把后压入堆栈的数据先弹出来才能保证恢复到原来的状态。在实际的程序设计中，由于每个应用程序的不同，还必须根据具体的情况来考虑是否需要保护？哪些数据需要保护等等，这就是单片机的堆栈为什么能够变化的原因。关于堆栈的操作先讲这些，后面的实验中我们还将结合具体的实验来分析，接下来再看另一种程序--综合程序的设计方法。

3．综合程序的设计方法

综合程序有查表程序、散转程序、数据排序程序、代码转换程序等等，作为初学者，要想全面的掌握也确实有一定的难度，所以只给大家简单地提一下，详细的内容就留到下则的课程中再来解释。

四．本课总结

程序设计是单片机开发最重要的工作，掌握程序设计的基本步骤和方法对于单片机的软件编写是至关重要的，这一课的内容较多，对于一时无法搞清的部分，大家可以结合以后的实际应用慢慢去理解，不要急于求成，千万记住一点，学习使用单片机绝不是一朝一夕的事，如果你不是天才，想速成单片机，我还是劝你赶紧改行！

五．第 16 课习题

1．什么是单片机的程序设计语言？
2．单片机的程序设计包括哪几个步骤？
3．画出单片机的流程图符号并简述它的作用？
4．单片机的分支结构程序指令有哪几条？
5．什么是单片机的程序嵌套？想想生活中还有哪些现象与单片机的嵌套类似。

第十七课 单片机的定时/计数器

通过前面十几节课的学习，我们已经掌握了很多的单片机知识，也许您已经可以用它来开发具体的产品了，不过在有些工业及民用控制中，我们往往需要定时检测某个参数或按一定的时间间隔来进行某项控制，比如家里的闹钟定时，电动机的 Y/Δ控制等等，此时您就要用到定时器或计数器，因此几乎所有的单片机系统内部都有几个定时/计数器，89C51 有两个 16 位的定时/计数器；而 89C52 则有 3 个。在了解这些定时/计数器之前，让我们先来熟悉几个基本概念：

一、几个基本概念

1．定时/计数的概念

从选票的统计谈起：画“正”，这就是计数，生活中计数的例子处处可见。例：线缆行业在电线生产出来之后要计米，也就是测量长度，怎么测量呢？用尺量？不现实，太长不说，要一边做一边量，怎么办呢？行业中有很巧妙的方法，用一个周长是 1 米的轮子，将电缆绕在上面一周，由线带动轮子转，这样轮子转一周不就是线长 1 米嘛，所以只要记下轮子转了多少圈，就可以知道走过的线有多少米了。

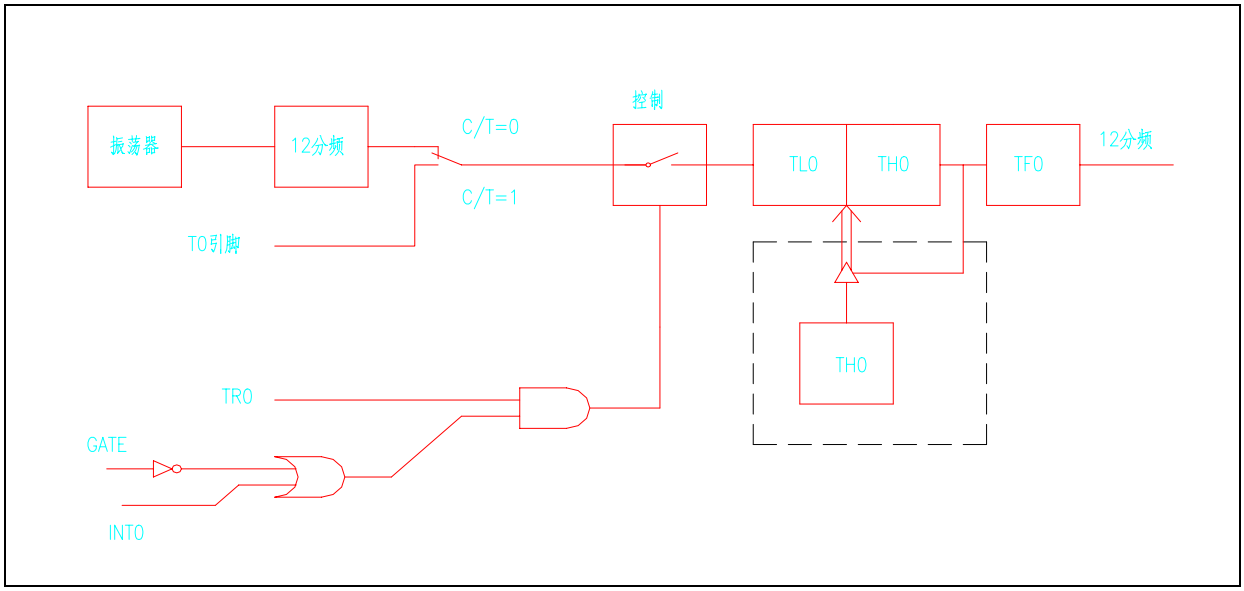
2．计数器的容量

从生活中的一个例子看起：一个水盆在水龙头下，水龙没关紧，水一滴一滴地滴入盆中。水滴不断落下，盆的容量是有限的，过一段时间之后，水就会逐渐变满，换句话说，计数是有容量的。那么单片机中的计数器有多大的容量呢？89C51 的两个计数器，分别称之为 T0 和 T1，这两个计数器都是由两个 8 位的 RAM 单元组成的，即每个计数器都是 16 位的计数器，最大的计数容量是 2^{16}=65536，记住是从 0-65535，因为在计算机中，往往把 0 作为起始点，比如 P0，P1.0，T0 等等。

3．定时器的原理

单片机中的计数器除了可以作为计数用，还可以用作定时器，定时器的用途当然很大，如闹钟的定时，手机的定时开关机等等，那么计数器是如何作为定时器来用的呢？一个闹钟，如果我们将它定时在 1 个小时后闹响，就相当于秒针走了 3600 次，在这里时间就转化成为了秒针走的次数，可见，计数的次数和时间之间的确有关，那么单片机的定时/计数器是怎么回事呢？请看下面的图：

从图中我们可以得出这样的结论：只要计数脉冲的间隔相等，那么计数值就代表了时间的流逝。其实，单片机中的定时器和计数器是一个东西，只不过计数器记录的是外界发生的事情，而定时器则是



由单片机提供一个非常稳定的计数源，然后把计数源的计数次数转化为定时器的时间，图中的 C/T 开关就是起这个作用的。那么提供给定时器的计数源又是从哪里来的呢？继续看上面的图，原来它就是由

单片机的晶振经过 12 分频后获得的一个脉冲源。我们知道，晶振的频率是很准的，所以这个计数脉冲的时间间隔当然也很准。

这里提个问题：一个 12M 的晶振，它提供给计数器的脉冲时间间隔是多少呢？不知大家是否还记得，那就是 1us，也就是 1 个微秒。

4．溢出的原理

继续让我们看水滴的例子，当盆中的水不断落下，最终会有一滴水使得盆中的水变满，这时如果再有一滴水落下，就会发生什么现象？水会溢出来，用个单片机的术语叫“溢出”。水溢出是流到地上，而计数器溢出后会使得 TF0 由“0”变为“1”，（至于 TF0 是什么我们稍后再谈），一旦 TF0 由“0”变为“1”，就是发生了变化，发生了变化就会引发事件，就象闹钟的定时时间一到，铃声就会响一样，那么单片机的溢出会引发什么事件呢？我们下节课再具体介绍，这里我们来研究另一个问题：要有多少个计数脉冲才会使 TF0 由“0”变为“1”。

5．任意定时及计数的方法

刚才已经讲过，51 系列单片机的计数器是 16 位的，也就是最大的计数值范围是 0-65535，因此计数器计到 65536 个脉冲就会产生溢出。这个不是问题，问题是我们在实际应用中经常会有少于 65536 个计数脉冲的要求，如药品生产线上，一箱为 50 瓶，一瓶为 100 粒，怎么样才能满足这个要求呢？

提示：如果是一个空的盆要 1 万滴水滴进去才会满，我们在开始滴水之前就预先放入一勺水，还需要 1 万滴嘛？单片机计数也是如此，如果我要计 5000 个脉冲，就先放进 60535 个，再来 5000 个，不就到了 65535 了吗？定时器同样如此，每个脉冲是 1 微秒，则计满 65536 个脉冲需时 65.536 毫秒，但现在我只要 10 毫秒就可以了，怎么办？10 个毫秒为 10000 微秒，所以，只要在计数器里预先放进 55536 就可以了，这种计数方法我们把它称之为预置数计数法。那么单片机的定时/计数器是由什么来控制的呢？下面就来讨论这个问题。

二．定时/计数器的方式控制字

从上一节我们已经知道，单片机中的定时/计数器可以有两种用途，那么我们怎样才能让它们工作于我们所需要的用途呢？这就需要通过定时/计数器的方式控制字（实际上就是与定时/计数器有关的特殊功能寄存器）来设置。在单片机中有两个特殊功能寄存器与定时/计数器有关，它们是 TMOD 和 TCON。顺便说一下，TMOD 和 TCON 是名称，我们在写程序时既可以直接用这个名称来指定它们，也可以直接用它们的地址 89H 和 88H 来指定它们（其实用名称也就是直接用地址，只不过汇编软件帮你翻译一下而已），具体使用稍后讲，现在先来看特殊功能寄存器 TMOD 的组成，看下表：

1．特殊功能寄存器 TMOD（89H）

用于 T1				用于 T0			
GATE	C/T	M1	M0	GATE	C/T	M1	M0

从表中可以看出，TMOD 被分成两部份，每部份 4 位，分别用于控制 T1 和 T0，至于这里面是什么意思，我们稍后介绍；先看另一个与定时/计数器有关的特殊功能寄存器 TCON。

2．特殊功能寄存器 TCON（88H）

用于定时/计数器				用于中断			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TCON 也被分成两部份，高 4 位用于定时/计数器，低 4 位则用于中断（我们暂时不管），而 TF0 我们前面已提到了，当计数溢出后 TF0 将由“0”变为“1”，原来 TF0 在这儿！那么 TR0、TR1 又是什么呢？看前面的图。计数脉冲要想进入计数器还真不容易，有层层关要通过，最起码，就是 TR0 要为“1”，开关才能合上，脉冲才能过来，因此，TR0 我们称之为运行控制位，当要使用 T0 时必须用指令 SETB 来置位以启动计数/定时器工作（用指令 CLR 可关闭定时/计数器的工作），这一切就在您的掌握中了。

知道了 TF0 和 TR0，TF1 和 TR1 的作用也清楚了。讲到这里，我们还是没有讲清楚单片机定时/计数器是如何来工作的，别着急，接着往下看。

三．定时/计数器的四种工作方式

单片机的定时/计数器共有四种工作方式，下面我们分别加以介绍：

1．工作方式 0

定时/计数器的工作方式 0 称之为 13 位定时/计数器方式。它由 TL（0/1）的低 5 位和 TH（0/1）的 8 位构成 13 位的计数器，此时 TL（0/1）的高 3 位未用。请看各位的功能：

（1）M1M0：

定时/计数器一共有四种工作方式，就是用 TMOD 的 M1M0 来控制的，两位正好是 4 种组合。

（2）C/T：

前面我们说过，定时/计数器既可作定时器用也可作计数器用，到底作什么用，由我们根据需要自行决定，也说是决定权在我们编程者手中。如果 C/T=0 就是用作定时器（开关往上打）；如果 C/T=1 就用作计数器（开关往下打）。顺便提一下：一个定时/计数器同一时刻要么作定时用，要么作计数用，不能同时使用，这一点请大家注意。

（3）GATE：

当我们选择了定时/计数器工作方式后，定时/计数脉冲却不一定能到达计数器端，中间还有一个开关，很显然如果这个开关不合上，计数脉冲就没法通过，那么开关什么时候合上呢？它有两种情况：

A．GATE=0，分析一下逻辑，GATE“非”后是“1”，进入“或”门，“或”门总是输出“1”，和“或”门的另一个输入端 INT0（中断 0，什么是中断，先不去管它）无关，在这种情况下，开关的打开、合上只取决于 TR0，只要 TR0=1，开关就合上，计数脉冲得以畅通无阻；而如果 TR0=0 则开关打开，计数脉冲无法通过，因此定时/计数是否工作，在这里只取决于 TR0。

B．GATE=1，在这种情况下，计数脉冲通路上的开关不仅要由 TR0 来控制，而且还要受到 INT0 引脚的控制，只有 TR0=1，且 INT0 也是高电平，开关才能合上，计数脉冲才得以通过。

那么为什么在这种模式下只用 13 位呢？干吗不用 16 位，这是为了和 51 的前辈 48 系列兼容而设的一种工作方式，如果你觉得用起来不顺手，那就干脆用工作方式 1 吧。

2．工作方式 1

工作方式 1 是 16 位的定时/计数器方式，将 TMOD 的 M1M0 设为“01”即可，其它特性与工作方式 0 相同，这里就不详细介绍了。

3．工作方式 2

在介绍这种工作方式之前先让我们思考一个问题：前面我们提到过任意计数及任意定时的问题，比如我要计 5000 个数，可是 16 位的计数器要计到 65535 才溢出，怎么办呢？我们讨论后得出的办法是采用预置数的办法--先在计数器里放上 60535 个，再来 5000 个脉冲，不就行了吗？是的，但是计满了之后我们又该怎么办呢？要知道，计数总是不断重复的，流水线上计满后马上又要开始下一次计数，下一次的计数还是 5000 吗？当计满并溢出后，计数器里面的值又变成了“0”（为什么，可以参考前面课程的说明），因此下一次将要计满 65535 后才会溢出，这可不符合要求，怎么办？当然办法很简单，就是每次一溢出时执行一段程序（这通常是需要的，要不然要溢出干吗？）可以在这段程序中做把预置数 60535 送入计数器中的工作。所以采用工作方式 0 或 1 都要在溢出后做一个重置预置数的工作，做工作当然就得要时间，一般来说这点时间不算什么，可是有一些场合我们还是要计较的，所以就有了工作方式 2--自动再装入预置数的工作方式。

既然要自动装入预置数，那么预置数就得放在一个地方，要不然装什么呢？那么预置数放在什么地方呢？它放在 T0（或 T1）的高 8 位中，那么这高 8 位不就不能参与计数了吗？是的，在工作方式 2 中，只有低 8 位参与计数，而高 8 位是不参与计数的，用作预置数的存放，这样计数范围就小了（当然做任何事总是有代价的，关键看值不值，如果我根本不需要计那么多数，那就可以用这种工作方式了）。看前面的图，每当计数溢出，就会打开 T0（或 T1）的高、低 8 位之间的开关，预置数就进入低 8 位。当然这是由硬件自动完成的，不需要我们去操心。

通常工作方式 2 用于波特率发生器（我们将在下册的串行接口中讲解），对于这种用途，定时器就是为了提供一个时间基准，计数溢出后不需做任何的事情，要做的仅仅只有一件，就是重新装入预置数，再开始计数，而且中间不能有任何的延迟，可见这个任务用这种工作方式来完成是最妙不过了。

4．工作方式 3

在这种工作方式下，T0 被拆成 2 个独立的定时/计数器来用。其中，TL0 可以构成 8 位的定时器或计数器工作方式；而 TH0 则只能作为定时器用，我们知道定时/计数器使用时需要有控制，计满后溢出需要有溢出标记，T0 被分成两个来用，那就要两套控制及溢出标记了，从何而来呢？TL0 还是用原来的 T0 的标记，而 TH0 则借用 T1 的标记，如此一来 T1 不是无标记、控制可用了吗？是的，在一般情况下，只有在 T1 以工作方式 2 运行时，才让 T0 工作于方式 3。

四．定时器/计数器的定时/计数范围

那么单片机的这四种工作方式的计数范围是如何确定的呢？

1．工作方式 0

13 位的定时/计数器工作方式。因此，最多可以计到 2 的 13 次方，也就是 8192 次。

2．工作方式 1

16 位的定时/计数器工作方式。因此，最多可以计到 2 的 16 次方，也就是 65536 次。

3．工作方式 2 和 3

工作方式 2 和工作方式 3 都是 8 位的定时/计数器工作方式，因此最多可以计到 2 的 8 次方，也说是 256 次。

预置值计算：用最大计数量减去需要的计数次数即可。例如：流水线上一个包装是 24 盒，要求每到 24 盒就产生一个动作，用单片机的工作方式 0 来控制，应当预置多大的值呢？对了，就是 8192-24=8168。

以上是计数器的原理，明白了这个道理，定时器也一样。前面的课程已经提到过，就不再重复了，请大家参考前面的例子自行分析。

本来想在这里做个实验，可有的问题可能还解释不清楚，所以等大家学完了下一课的中断后一块儿来实验吧，请继续把下一课的内容看完。

五．本课总结

本课主要讲述了单片机的定时/计数器的原理和工作方式。定时/计数器是单片机的重要组成部分，如果您在用单片机开发产品时需要用到定时器或计数器，那么本课的内容是必须掌握的，所以希望大家能认真地看一下这一课的内容，后面我们还会结合具体的实验来讲解它们的使用方法。

六．第 17 课习题

1．89C51 中有几个定时/计数器？它们的计数范围是多少？

2．什么是单片机的溢出？溢出后会产生什么现象？

3．定时/计数器有关的两个 SFR 是什么？它们的地址为什么？

4．单片机有几种工作方式？它们的定时/计数范围是多少？

第十八课 单片机的中断系统

由于不讲中断无法解释实验的某些结果，所以就等这一课讲完了再一起做实验，我们先来看单片机的中断系统原理。

单片机中断系统的目的是为了让 CPU 对内部或外部的突发事件及时地作出响应，并执行相应的程序，在单片机的开发中，它有着十分重要的作用，那么单片机的中断是怎么回事？它是如何工作的呢？这一课就来讨论这个问题，在讲解之前让我们先来介绍一下中断的原理：

一、中断的基本原理

什么是中断？中断的过程是什么？要搞清楚这个问题，我们同样先从生活中的一个例子开始：你正在家中看书，突然电话铃响了，你放下书，去接电话，和来电话的人交谈，通完电话，回来继续看你的书，这就是生活中的“中断”现象—就是正常的工作过程被外部的事件打断了。仔细研究一下生活中的中断，对我们学习单片机中断会很有帮助：

第一，中断源。什么可引起中断，生活中很多事件都可以引起中断，比如：有人按了门铃、电话铃响了、你的闹钟响了、你烧的水开了……等等诸如此类的事件，我们把可以引起中断的事件称之为中断源。单片机中也有一些可以引起中断的事件（比如：按下键盘、定时/计数器溢出、报警等等），89C51 单片机中共有 5 个中断源：两个外部中断，两个定时/计数器中断和一个串行口中断。

第二，中断的嵌套与优先级处理。设想一下，我们正在看书，电话铃响了，同时又有人按了门铃，你先做那样呢？如果你正在等一个很重要的电话，一般是不会去理会门铃的；而反之，如果你正在等一个很重要的客人，则可能就不会去理会电话了；如果两者都不是（既不等电话，也不等人上门），你可能会按你通常的习惯去处理。总之这里存在一个优先级的问题，单片机中也是如此，也有优先级的问题。优先级的问题不仅仅发生在两个中断同时产生的情况，也发生在一个中断已产生，又有另一个中断产生的情况。比如你正在接电话，又有人按门铃的情况，或者你正开门与人交谈，又有电话响了的情况。仔细想一下，我们一般会怎么处理？

不会吧，这样就有点手忙脚乱了？要是再来个 MM 这么办，呵呵！

第三，中断的响应与处理。当有事件产生，进入中断之前我们必须先记住现在的书看到第几页了，或拿一个书签放在当前页的位置，然后去处理不同的事情（因为处理完了，我们还要回来继续看书），电话铃响我们要到放电话的地方去，门铃响了我们要到门那边去，也就是说不同的中断，我们要在不同的地点处理，而这个地点通常还是固定的。单片机中采用的也是这种方法，五个中断源，每个中断产生后都要到一个固定的地址去找处理这个中断的程序，当然在去之前首先要保存下面将执行的指令的地址，以便处理完中断后回到原来的地方继续往下执行程序。具体地说，单片机中断响应可以分为以下几个步骤：1、停止主程序运行。当前指令执行完后立即终止现在执行的程序。2、保护断点。把程序计数器 PC 的当前值压入堆栈，保存终止的地址（即断点地址），以便从中断服务程序返回时能够继续执行该程序，3、寻找中断入口。根据 5 个不同的中断源所产生的中断，查找 5 个不同的入口地址。4、执行中断处理程序。这就不讲了；5、中断返回。执行完中断处理程序后，就从中断处返回到主程序，继续往下执行。以上工作是由计算机自动完成的，与编程者无关，在这 5 个入口地址处存放有中断处理的程序（这是程序编写时放在那儿的，如果没把中断处理程序放在那儿可就错了，因为中断程序无法被执行到）。那么执行中断程序有什么好处呢？

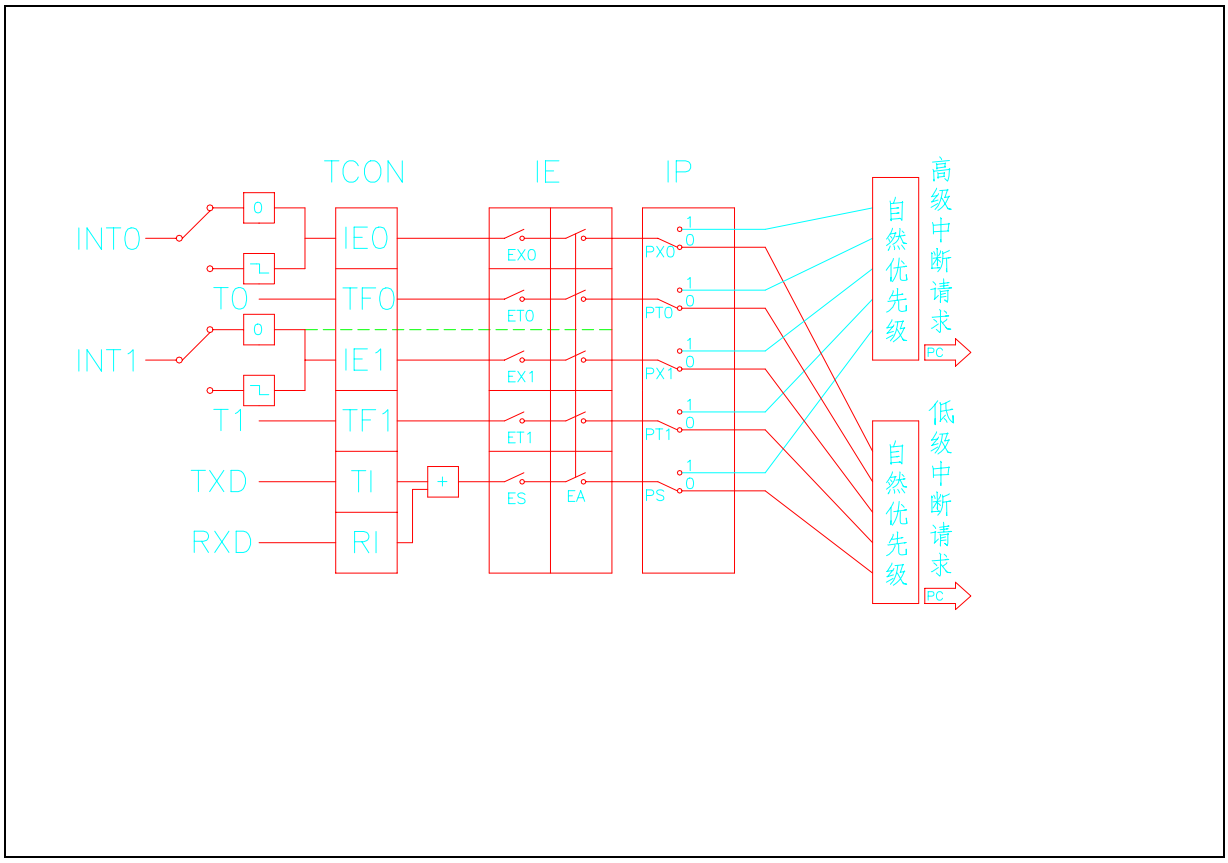
二、实现中断的好处

单片机为什么要有中断系统，使用中断有什么好处呢？日常生活中，我们除了看书，肯定还要做很多其他的事情，比如听电话，接待客人，烧水吃饭等等，单片机实行中断也有很多的好处，具体来说：1。实行分时操作，提高 CPU 的效率。只有当服务对象向 CPU 发出中断申请时，才去为它服务，这样我们就可以利用中断功能同时为多个对象服务，从而大大提高了 CPU 的工作效率。2。实现实时处理。利用中断技术，各个服务对象可以根据需要随时向 CPU 发出中断申请，及时发现和处理中断请求并为

之服务，以满足实时控制的要求，比如定时的时间到了，就要 CPU 做相应的处理。3。进行故障处理。对难以预料的情况或故障，比如掉电，事故等，可以向 CPU 发出请求中断，由 CPU 作出相应的处理。那么单片机是如何实现中断处理的呢？要了解这个问题，就让我们先来看看单片机中断系统的内部结构。

三．单片机中断系统的结构：

前面已经提到，89C51 单片机有 5 个中断源，看下面的图，这就是 89C51 单片机的中断系统内部结构图，让我们一一来进行分析：



1．中断源：

（1）外部中断：

即外中断 0 和外中断 1，经由外部引脚引入，在单片机的硬件上有两个引脚（12 脚和 13 脚），名称为 INT0 和 INT1（第二引脚功能 P3.2、P3.3）。在单片机的内部有一个特殊功能寄存器 TCON，其中有四位是与外中断有关的。还记得 TCON 是什么吗？对了，是定时器控制寄存器，请看下面的表：

用于定时/计数器				用于中断			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

A． IT0：中断 0（INT0）的触发方式控制位。

可由软件进行置位和复位，IT0=0，中断 0 为低电平触发方式；IT0=1，中断 0 为负跳变触发方式。这两种方式的差异讲起来有点复杂，这里就不介绍了，作为初学者，只要知道就可以了。

B．IE0：中断 0（ INT0）的中断请求标志位。

当有外部的中断请求时，该位就会置“1”；在 CPU 响应中断后，该位就自动清“0”。（注意☺：这是由硬件自动完成的）。

IT1、IE1 的用途和 IT0、IE0 是类似的。看上面的表：

（2）内部中断

即定时器 0（T0）和定时器 1（T1）中断，与外中断一样，它也是由 TCON 中的四位控制的。

TF0：定时器 T0 的溢出中断标记。当 T0 计数器产生溢出时，由硬件置位 TF0；当 CPU 响应

中断后，再由硬件将 TF0 自动清“0”。

TF1：与 TF0 类似，这里就不讲了。

（3）串行口中断

负责串行口的发送、接收中断，具体内容我们到下册学习串行接口时再详细讲解。下面我们再来讲另一个与中断有关的寄存器。

2、中断允许寄存器 IE（A8H）

中断的允许或禁止是由片内可进行位（什么是位，大家可别到现在还说不知道哦）寻址的 8 位中断允许寄存器 IE 来控制的，允许中断我们把它称为中断开放，不允许中断我们把它称为中断屏蔽），如何操作，说穿了其实很简单，就是通过对 IE 的相应位的置“1”或请“0”就可以了，请看下面的表格：

中断允许寄存器 IE							
EA	×	×	ES	ET1	EX1	ET0	EX0

（1）EA：总中断允许开关。它是个总开关，凡是要设置中断都得先通过它。EA=1，开放所有的中断；EA=0，则所有中断都被禁止。

（2）ES：串行口中断控制位。ES=1，允许中断；ES=0，禁止中断。

（3）ET1：定时/计数器 1 中断控制位。ET1=1，允许中断；ET1=0，禁止中断。

（4）EX1：外中断 1 中断控制位。EX1=1，允许中断；EX1=0，禁止中断。

（5）ET0：定时器 0 中断控制位。ET0=1，允许中断；ET0=0，禁止中断。

（6）EX0：外中断 0 中断控制位。EX0=1，允许中断；EX0=0，禁止中断。

例如：我们现在要设置 T1 允许，INT1 允许，其它不允许，则 IE 应该是 10001100（即 8CH），也可以直接用位操作指令 SETB　EA；SETB　ET1；SETB　EX1 来实现它。看一下下面的表：

EA	×	×	ES	ET1	EX1	ET0	EX0
1	0	0	0	1	1	0	0

这里有一点请大家注意☺：当复位 CPU 时，IE 将被全部清“0”。

了解了中断的设置，让我们再来看另一个问题：前面我们提到过，中断有优先级和嵌套的问题，那么中断的优先级和嵌套是如何来控制的呢？接着往下看：

3．中断源优先级寄存器 IP（D8H）

单片机执行中断的过程和生活中的中断有些类似，它也有一个自然优先级与人工优先级的问题，那么单片机是如何来设置它们的呢？这就要用到中断优先级寄存器 IP，它也是一个可位寻址的 8 位寄存器。现在让我们先来看五个中断源的自然优先级是如何设置的：

五个中断源的自然优先级由高到低的排列顺序为外中断 0→定时器 0→外中断 1→定时器 1→串口中断。如果我们不对其进行设置，单片机就按照此顺序不断的循环检查各个中断标志（就像我们生活中按照习惯处理事物一样），但有时我们需要人工设置高、低优先级，也就是说由编程者来设定哪些中断是高优先级、哪些中断是低优先级（当然由于只有两级，所以必然只有一些中断处于优先级别，而其他的中断则处于同一级别，处于同一级别的中断顺序就由自然优先级来确定，这一点请大家务必搞清楚了）。

既然可以设定人工优先级，那么它又是如何来设置的呢？其实很简单，我们只要把 IP 寄存器的对应位置“1”就可以了，看下面的表：

×	×	×	PS	PT1	PX1	PT0	PX0
			串口	T1	INT1	T0	INT0

开机时，每个中断都处于低优先级，我们可以用指令来对优先级进行设置。例如：现在有如下要求，将 T0、INT1 设为高优先级，其它为低优先级，求 IP 的值。

IP 的首 3 位没用，可任意取值，设为 000，后面根据要求写：00000110，即 IP=06H，看下面的表。

×	×	×	PS	PT1	PX1	PT0	PX0
0	0	0	0	0	1	1	0

这里有个问题：如果 5 个中断请求同时发生时又会出现什么情况呢？比如在上例中，五个中断同时发生，求中断响应的次序。按照我们学到的内容，响应次序应该为：定时器 0→外中断 1→外中断 0

→实时器 1→串行中断。是不是符合我们刚才说的除了人工设置的高优先级外，其余的均按照自然优先级来处理。其实这很好理解，如果我在家等待一个很重要的电话，同时又有人来敲门或者烧的水开了，当我放下电话后，还是会按照一般的习惯去处理其他的事情（比如先开门让客人进来，再去处理烧开的水）。

4．串行口控制寄存器 SCON（98H）

用于串行口中断及控制，我们留到下册中再详细解释。下面讨论一下另一个问题：

四．中断响应的条件和过程

1、中断响应的原理

讲到这里，我们依然对于单片机响应中断的过程感到神秘，我们人响应外界的事件，是因为我们有多种“传感器”——眼、耳等可以接受不同的信息，那么单片机是如何做到这点的呢？其实说穿了，一点也不稀奇，单片机工作时，在每个机器周期中都去查询一下各个中断标记，看它们是否是“1”，如果是“1”，就说明有中断请求了。所以所谓中断，其实也就是查询，只不过是每个周期都查一下而已。这要换成人来说，就相当于你在看书的时候，每一秒钟都抬起头来看是不是有人按门铃了，是否有电话来了，烧的水是否开了……很蠢，是吗？本来嘛，计算机它根本就没人聪明。了解了响应中断的原理，就不难理解中断响应的条件了。

2、中断响应的条件

当有下列三种情况之一发生时，CPU 将封锁对中断的响应，而是到下一个机器周期时再继续查询：

（1）CPU 正在处理一个同级或更高级别的中断请求时。

（2）当前的指令没有执行完时。

我们知道，单片机有单周期指令、双周期指令、三周期指令和四周期指令，如果当前执行指令是单周期指令也许没关系，如果是双周期或四周期指令，那就要等整条指令都执行完了，才能响应中断（因为中断查询是在每个机器周期都可能查到的）。

（3）当前正执行的指令是返回指令（RET 或 RETI）或访问 IP、IE 寄存器的指令，则 CPU 将至少再执行一条指令才能响应中断。

这些都是与中断有关的寄存器，如果正访问 IP、IE 则可能会出现开、关中断或改变中断的优先级；而中断返回指令则说明本次中断还没有处理完，所以就要等本指令处理结束，再执行一条指令才可以响应中断。

3、中断响应的过程：

CPU 响应中断时，首先把当前指令的下一条指令（就是中断返回后将要执行的指令）的地址（断点地址）送入堆栈，然后根据中断标记，硬件执行跳转指令，转到相应的中断源入口处，执行中断服务程序，当遇到 RETI（中断返回指令）时，返回到断点处继续执行程序，这些工作都是由硬件自动来完成的。那么中断入口的地址是如何来确定的呢？在 51 系列单片机中，五个中断源都有它们各自的中断入口地址，请看下面：

（1） 外中断 0（INT0）：0003H

（2） 定时器 0（T0） ：000BH

（3） 外中断 1（INT1）：0013H

（4） 定时器 1（T1） ：001BH

（5） 串口中断 ：0023H

讲到这里，大家应该明白，为什么我们前面的有些程序一开始是这样写的：

```
ORG   0000H   ;
LJMP  START   ;
ORG   0030H   ;
START: *****  ;
       *****  ;
       *****  ;
```

```
END。
```

其实这样写的目的，就是为了让出中断源所占用的地址，当然，在程序中如果没有用到中断时，直接从地址 0000H 开始写理论上不是不可以，但在实际工作中最好不要这样做。这里还有一个问题，大家是否注意到，每个中断向量地址只间隔了 8 个字节，如 0003H－000BH，在如此少的空间中如果完成不了中断程序，又该这么办呢？其实很简单，您只要在中断处安排一条 LJMP 指令，不就可以把中断服务程序跳转到任何地方去了吗？所以一个完整的主程序看起来应该是这样的：

```
ORG    0000H      ;
LJMP   START      ;
ORG    0003H      ;
LJMP   AINT0      ；转外中断 0 服务程序
ORG    000BH      ;

START：*****   ；主程序开始
       *****   ;
       *****   ;

AINT0：*****   ；中断服务程序
       *****   ;
  RETI          ；从中断服务程序返回
END。
```

中断程序处理完成后，一定要执行一条 RETI 指令，执行这条指令后，CPU 将会把堆栈中保存着的断点地址取出，送回程序计数器 PC 中，那么程序就会根据 PC 中的值从主程序的中断处继续往下执行了。从 CPU 终止当前程序，且转向另一程序这点看，中断的过程很象子程序，其实它们之间还是有区别的：中断发生的时间是随机的，而子程序调用则是按程序进行的，所以它们的返回命令也是不一样的。RET 是用在返回子程序中的，而 RETI 则用在返回中断处理程序中的，这一点千万不能搞错了。

五．本课总结

本课主要讲述中断的工作原理和响应的过程以及如何来设置与中断有关的寄存器，中断是单片机响应外界事件的重要组成部分，与定时/计数器一样，掌握它的工作原理对我们使用单片机开发产品是非常必需的。

六．第 18 课习题

1．51 单片机有哪几个中断源？它们的名称分别是什么？
2．中断请求源是由哪些寄存器控制的？
3．中断响应的过程是什么？
4．简述 RET 和 RETI 的区别。

第十九课 定时与中断实验（一）

前面的两节课程我们连续的介绍了单片机定时/计数器和中断的原理及结构，可能大家不是很理解，这节课我们就来做几个实验验证一下前面所学的内容。

实验一、利用定时器实现灯的闪烁

在开始学单片机时我们所做的第一个实验就是 LED 灯的闪烁，不过那是用延时程序做的，现在回想起来，这样做不是很恰当，为什么呢？因为我们的主程序做了灯的闪烁，就不能再干其它的事情了，难道单片机只能这样工作吗？当然不是，我们可以用定时器来完成灯的闪烁功能。

程序如下：

```
ORG    0000H                ;
AJMP    START               ;
ORG    30H                  ;
START:MOV    P1,#0FFH    ;关所有的灯
MOV    TMOD,#00000001B ;定时/计数器 0 工作于方式 1
MOV    TH0,#15H            ;
MOV    TL0,#0A0H           ;立即数（15AH+0A0H=5536）
SETB    TR0                ; 定时/计数器 0 开始运行
LOOP:JBC    TF0,NEXT     ;如果 TF0 等于 1，则清 TF0 并转 NEXT 处
此处可以加入其他的任何指令
AJMP    LOOP               ;否则跳转到 LOOP 处运行
NEXT:CPL    P1.0           ;
MOV    TH0,#15H            ;
MOV    TL0,#0A0H            ;重置定时/计数器的初值
AJMP    LOOP               ;
END。
```

把程序下载到实验板，看到了什么？灯开始闪烁了，这可是用定时器做的，不再是主程序的循环了。简单分析一下程序，为什么用 JBC 呢？TF0 是定时/计数器 0 的溢出标记位，当定时器产生溢出后，该位由“0”变“1”，所以查询该位就可知道定时时间是否已到，该位为“1”后，要用软件将标记位清“0”，以便下一次定时时间到了将该位由“0”变为“1”，所以用了 JBC 指令，该指令前面已经学过--判“1”转移的同时，将该位清“0”。

以上程序可以实现灯的闪烁了，可是主程序除了让灯闪烁外，还是不能做其他的事啊！不对，我们可以在 LOOP：JBC　TF0，NEXT 和 AJMP　LOOP 指令之间插入一些指令来做其他的事情，只要保证执行这些指令的执行时间少于定时时间就可以了。

当然，这样的方法还不是最好，所以我们常用下面的方法实现延时程序：

实验二、利用中断方式实现灯的延时

程序如下：

```
ORG    0000H          ;
AJMP    START         ;
ORG    000BH           ;定时器 0 的中断向量地址
AJMP    TIME0          ;跳转到真正的定时器程序处
ORG    30H             ;
START:MOV    P1,#0FFH    ;关所有灯
MOV    TMOD,#01H        ;定时/计数器 0 工作于方式 1
```

```
MOV   TH0,#15H          ;
MOV   TL0,#0A0H         ;立即数 5536
SETB  EA                ;开总中断允许
SETB  ET0               ;开定时/计数器 0 允许
SETB  TR0               ;定时/计数器 0 开始运行
LOOP:AJMP  LOOP         ;真正工作时，这里可写任意程序
TIME0:
PUSH  ACC               ;将 ACC 推入堆栈保护
PUSH  PSW               ;将 PSW 推入堆栈保护
CPL   P1.0              ;取反 P1.0
MOV   TH0,#15H          ;
MOV   TL0,#0A0H         ;重置定时常数
POP   PSW               ;
POP   ACC               ;
RETI                    ;
END。
```

上面的例子中，定时时间一到，TF0 由“0”变“1”，就会引发中断，CPU 将自动转至 000BH 处寻找程序并执行，由于留给定时器中断的地址空间只有 8 个字节，显然不足以写下所有的中断处理程序，所以在 ORG 000BH 后安排了一条长跳转指令，转到实际处理中断的程序处，这样，中断程序可以写在任意地方，也可以写任意长度了。单片机进入定时中断后，首先要保存当前的一些状态，在这里程序只演示了保存 ACC 和 PSW，实际工作中应该根据需要将可能会改变的单元的值都推入堆栈进行保护（本程序中实际不需保护任何值，这里只作个演示）。

这是一项很重要的工作，因为 CPU 所做的自动保护工作是很有限的，它只保护了一个地址（就是中断返回后将要执行的指令的地址，也叫断点地址），而其它的所有东西都不保护，所以如果你在主程序中用到了如 ACC、PSW 等寄存器，而在中断程序中又要用到它们，还要保证回到主程序后这里面的数据还是没执行中断之前的数据，就得自己把它们保护起来。否则程序执行的结果就不是您想象的要求了！

上面的两个程序运行后，我们发现灯的闪烁非常快，根本分辨不出来，只是视觉上感到有些晃动而已，为什么呢？我们可以计算一下，定时器中预置的数是 5536，所以每计 60000（65536-5536）个脉冲就是定时时间到，这 60000 个脉冲的时间是多少呢？我们的晶振是 12 兆的，所以就是 60000 微秒，即 60 毫秒，因此速度是非常快的。如果我想实现一个 1 秒的定时器，该怎么办呢？在该晶振频率下，最长的定时也就是 65.536 个毫秒啊！请看第三个实验：

实验三、延长定时时间的方法

程序如下：

```
ORG   0000H             ;
AJMP  START             ;
ORG   000BH             ;定时器 0 的中断向量地址
AJMP  TIME0             ;跳转到真正的定时器程序处
ORG   30H               ;
START:MOV  P1,#0FFH ;关所有的灯
MOV   30H,#00H          ;软件计数器预清 0
MOV   TMOD,#01H         ;定时/计数器 0 工作于方式 1
MOV   TH0,#3CH          ;
MOV   TL0,#0B0H         ;立即数（3CH+0BH=15536）
SETB  EA                ;开总中断允许
```

```
SETB   ET0                ;开定时/计数器 0 允许
SETB   TR0                ;定时/计数器 0 开始运行
LOOP:AJMP   LOOP          ;真正工作时,这里可写任意程序
TIME0:
PUSH   ACC                ;将 ACC 推入堆栈保护
PUSH   PSW                ;将 PSW 推入堆栈保护
INC   30H                  ;
MOV   A,30H                ;
CJNE   A,#20,TIME1         ;30H 单元中的值到了 20 了吗?
CPL   P1.0                 ;到了，取反 P1.0
MOV   30H,#0               ;清软件计数器
TIME1:MOV   TH0,#15H       ;给 T0 重新赋值
MOV   TL0,#9FH             ;重置定时常数
POP   PSW                  ;
POP   ACC                  ;
RETI                       ;
END。
```

先自己分析一下程序，看看是怎么实现的？这里采用了软件计数器的概念，思路是这样的，先用 T0 做一个 50 毫秒的定时器，定时时间到了之后并不是立即取反 P1.0，而是将软件计数器中的值加 1，如果软件计数器计到了 20，就取反一次 P1.0，并清掉软件计数器中的值；否则直接返回。这样，就变成了 20 次定时中断才取反一次 P1.0，因此定时时间就延长了成了 20*50mS，即 1000 毫秒了。这个思路在实际的工程应用中是非常有用的，比如有时候我们需要若干个定时器，可 89C51 中总共才有 2 个，怎么办呢？其实，只要这几个定时器在时间上有一定的公约数，我们就可以用软件定时器加以实现。例如要实现 P1.0 口所接灯按 1S/次，而 P1.1 口所接灯按 2S/次闪烁，怎么实现呢？我们可以用两个计数器，一个在它计到 20 时，取反一次 P1.0，并清“0”，如上面所示；而另一个则计到 40 取反一次 P1.1，然后清“0”，不就行了吗？看下面的实验：

实验四、软件定时器的实现方法

程序如下：

```
ORG   0000H      ;
AJMP   START    ;
ORG   000BH     ;定时器 0 的中断向量地址
AJMP   TIME0    ;跳转到真正的定时器程序处
ORG   0030H     ;
START:MOV   P1,#0FFH ;关所有的灯
MOV   30H,#00H          ;软件计数器预清 0
MOV   TMOD,#01H         ;定时/计数器 0 工作于方式 1
MOV   TH0,#3CH           ;
MOV   TL0,#0B0H          ;立即数 15536
SETB   EA                ;开总中断允许
SETB   ET0               ;开定时/计数器 0 允许
SETB   TR0               ;定时/计数器 0 开始运行
LOOP:AJMP   LOOP         ;真正工作时,这里可写任意程序
TIME0:
PUSH   ACC                ;将 ACC 推入堆栈保护
PUSH   PSW                ;将 PSW 推入堆栈保护
```

```
INC   30H                    ;
INC   31H                    ;两个计数器都加 1
MOV   A,30H                  ;
CJNE  A,#20,TNEXT            ;30H 单元中的值到了 20 了吗
CPL   P1.0                   ;到了，取反 P1.0
MOV   30H,#0                 ;清软件计数器
TNEXT:MOV  A,31H             ;
CJNE  A,#40,TRET             ;31H 单元中的值到 40 了吗
CPL   P1.1                   ;
MOV   31H,#0                 ;到了，取反 P1.1 并清计数器，返回
TRET:MOV  TH0,#15H           ;
MOV   TL0,#9FH               ;重置定时常数
POP   PSW                    ;
POP   ACC                    ;
RETI                         ;
END。
```

这就是软件定时器的用法，试试看，用软件定时器做一个交通信号灯的实验，要求红灯亮 2 分钟，黄灯亮 1 分钟，绿灯亮 3 分钟，然后红灯再亮 2 分钟，黄灯再亮 1 分钟，绿灯再亮 3 分钟，不断的循环……。通过上面的几个实验，大家对定时器的使用方法是不是有了一个初步认识，接下来让我们共同来做一个更为有趣的实验：用单片机做一个乐曲编奏器。

实验五、可编程乐曲演奏器

单片机用作乐曲编奏器的原理是，通过控制定时器的定时时间来产生不同频率的方波，驱动喇叭发出不同音阶的声音，再利用延迟来控制发音时间的长短，就能控制音调中的节拍。编程时，把乐谱中的音符和相应的节拍变换为定时常数和延迟常数，把数据表格存放在存储器中，由查表程序得到相应的定时常数和延迟常数，分别用定时器控制产生方波的频率和发出该频率方波的持续时间，当延迟时间到了之后，再查下一个音符的定时常数和延迟常数。依次进行下去，就可自动演奏出悦耳动听的乐曲来。

程序如下：

```
ORG   001BH    ;定时器 T1 的中断入口
MOV   TH1,R1   ;重装定时初值
MOV   TL1,R0   ;
CPL   P3.7     ;P1.0 输出方波
RETI           ;中断返回
ORG   100H     ;主程序
START:MOV  TMOD,#01H ;定时器 T1 工作方式 1
MOV   IE,#88H        ;允许 T1 中断
MOV   DPTR,#TAB      ;表格首地址
LOOP:CLR   A         ;
MOVC  A,@A+DPTR      ;查表
MOV   R1,A           ;定时器高 8 为存 R1
INC   DPTR           ;
CLR   A              ;
MOVC  A,@A+DPTR      ;查表
MOV   R0,A           ;定时器低 8 为存 R0
ORL   A,R1           ;
JZ    NEXT0          ;全 0 为休止符
```

```
MOV    A,R0                ;
ANL    A,R1                ;
CJNE   A,#0FFH,NEXT        ;全 1 表示乐曲结束
SJMP   START               ;从头开始，循环演奏
NEXT:MOV   TH1,R1          ;装入定时值
MOV    TL1,R0              ;
SETB   TR1                 ;启动定时器
SJMP   NEXT1               ;
NEXT0:CLR  TR1             ;关闭定时器，停止发音
NEXT1:CLR  A               ;
INC    DPTR                ;
MOVC   A,@A+DPTR           ;查延迟常数
MOV    R2,A                ;
LOOP1:LCALL   D200         ;调用延时 200mS 子程序
DJNZ   R2,LOOP1            ;控制延迟次数
INC    DPTR                ;
AJMP   LOOP                ;处理下一个音符
D200:MOV   R4,#81H         ;延时 20mS 子程序
D200B:MOV  A,#0FFH         ;
D200A:DEC  A               ;
JNZ    D200A               ;
DEC    R4                  ;
CJNE   R4,#00H,D200B       ;
RET                        ;
TAB:       DB   0FEH,25H,02H,0FEH,25H,02H;     DB   0FEH,84H,02H,0FEH,84H,02H;
           DB   0FEH,84H,04H,0FEH,25H,04H;     DB   0FEH,25H,02H,0FEH,84H,02H;
           DB   0FEH,0C0H,04H,0FEH,0C0H,04H;   DB   0FEH,98H,02H,0FEH,84H,02H;
           DB   0FEH,57H,08H,00H,00H,04H;      DB   0FFH,0FFH;
           END。
```

上述程序是歌曲“新年好”的一段简谱：1=C　　1 1 1 5 | 3 3 3 1 | 1 3 5 5 | 4 3 2 — |，如何来实现的呢？。单片机的晶振为 12M，则乐曲、频率及定时常数三者之间的对应关系如下表所示：

C 调音符	5	6	7	1	2	3	4	5	6	7
频率（Hz）	392	440	494	524	588	660	698	784	880	988
半周期 mS	1.28	1.14	1.01	0.95	0.85	0.76	0.72	0.64	0.57	0.51
定时值	FD80	FDC6	FE07	FE25	FE57	FE84	FE98	FEC0	FEE3	FF01

我们用定时器 T1 工作方式 1 来产生歌谱中各音符对应频率的方波，由 P3.7 输出驱动蜂鸣器，节拍的控制通过调用延时子程序 D200（延时 200mS）的次数来实现，以每拍 800mS 的节拍时间为例，一拍需要调用 D200 子程序 4 次，同样的道理，半拍就需要调用 2 次，由此就演奏出了上面的乐曲。

这里给大家讲明一点：上面的程序是我从其他的单片机书籍上照搬过来的，由于我对音乐实在是一窍不通，所以上面的程序我也不知道是否正确，还请大家自己分析一下，如果有不对的地方请您多多包涵!!!

六．本课总结

本课主要介绍了定时/计数器的使用方法及编程技巧，大家可以通过反复实验来加强和巩固上两节课所学的知识。另外请您结合第十六课的内容把这几段程序的流程图画出来，这就算是本课的习题吧。

第二十课 定时与中断实验（二）

前面我们用定时器和中断做了延时的实验，现在再来看一看外部计数和外部中断的实验，在实际的工程应用中计数器通常会有两种要求：第一种，将计数的值显示出来，象录音机上的计数器、汽车上的里程表等等；第二种，计数值到一定值后即中断报警，如前面提到的生产线上的计数、定长定量仪等等。

接下来我们先来做一个外部计数器的实验。要将外部计数的值显示出来，最好是用数码管，可我们还没有讲到这一部份，为了避免把问题复杂化，我们用 P1 口的 8 个 LED 来显示计到的数据。为了完成这个实验，我们需要用到一套附件，它的作用就是输出一个宽度为 500mS 的方波，这套附件不在我们的实验板上。为了节约大家的学习费用，我特地做了几套供大家借用，各位可以到我这儿来借，免费使用，不过请各位爱惜哦，不要搞坏了。有了这套附件就可以完成我们的实验了。我们把附件的两根线分别连接到实验板的电源接口和单片机的 15 脚（也就是 T1 的输入端）。实验的程序如下：

一．计数器实验一

程序如下：

```
ORG   0000H     ;
AJMP   START   ;
ORG   0030H     ;
START:MOV   SP,#5FH   ;
   MOV   TMOD,#40H    ;定时/计数器 1 作计数用，T0 不用全置“0”
   SETB   TR1            ;启动 T1 开始运行
LOOP:MOV   A,TL0     ;
   MOV   P1,A          ;
   AJMP   LOOP        ;
END。
```

运行这个程序，看到了什么？随着 LED 的闪烁，实验板上的 8 个 LED 也在不断地变化，注意观察，是不是按二进制：00000000，00000001，00000010，00000011，……这样的顺序在变呢？对了，这就是 TL0 中的数据。不过这个实验还看不出什么名堂，接着做第二个实验：

二．计数器实验二

程序如下：

```
ORG   0000H       ;
AJMP   START      ;
ORG   001BH       ;
AJMP   TIMER1     ;定时器 1 的中断处理
ORG   0030H       ;
START:MOV   SP,#5FH ;
   MOV   TMOD,#40H ;定时/计数器 1 作计数用，工作方式 1，T0 不用置“0”
   MOV   TH1,#0FFH   ;
   MOV   TL1,#0FAH   ;预置值，每计到 6 个脉冲即为一个事件
   SETB   EA          ;
   SETB   ET1         ;开总中断和定时器 1 中断允许
   SETB   TR1         ;启动定时/计数器 1 开始运行
   AJMP   $           ;
TIMER1:PUSH   ACC     ;
```

```
    PUSH   PSW              ;
    CPL    P1.0             ;计数值到，即取反 P1.0
    MOV    TH1,#0FFH        ;
    MOV    TL1,#0FAH        ;重置计数初值
    POP    PSW              ;
    POP    ACC              ;
    RETI                    ;
    END。
```

这段程序完成的工作其实很简单，就是每 6 个计数脉冲到来后取反一次 P1.0，因此实验的结果应当是：15 脚接的 LED 亮、灭 6 次，则 P1.0 口所接的 LED 就亮或灭一次，这就是我们对输入脉冲的计数，也就是每 6 个计数产生一次中断。这段程序中有一个符号以前没见过，需要给大家解释一下，AJMP $，“$”我们称为标识符，它的作用是指这条指令的开始处，在这里，其实就是循环执行 AJMP $这条指令，执行这么多次干什么，实际上它是在等待中断的产生。

这两个实验需要附件，如果您觉得做起来不大方便，没关系，我们接着来做第三个实验，采用两个定时/计数器合用，一个作为定时器用，一个作为计数器用，来实现 P1.1 的延时，这可以直接在我们的实验板上完成。

三．两个定时/计数器合用的延时实验

采用两个定时/计数器，其中 T0 作为定时器用，工作方式为方式 1，T1 作为计数器用，计数次数为 1000 次。T0 溢出时，产生一个间隔为 60mS 的方波（也就是让 LED3 各亮灭 60mS），然后把 P1.2 的输出作为 T1 的计数脉冲，T1 计数溢出时（满 1000 次），取反一次 P1.1，产生一个周期为 2 秒的方波（即 LED2 每 2 秒闪烁一次）。

程序如下：

```
ORG    0000H     ;
AJMP   MAIN      ;
ORG    000BH     ;定时器 T0 的中断入口
AJMP   T_0       ;转 T0 中断服务程序
ORG    001BH     ;定时器 T1 的中断入口
AJMP   T_1       ;转 T1 中断服务程序
ORG    0030H     ;
MAIN:MOV   TMOD,#51H ;T1 为计数器方式 1，T0 为定时器方式 1
   MOV    TH0,#15H       ;设置 T0 初值
   MOV    TL0,#0A0H      ;
   MOV    TH1,#0FCH      ;设置 T1 初值
   MOV    TL1,#18H       ;
   MOV    IE,#8AH        ;允许     T0、T1 中断
   SETB   TR0            ;启动定时器 T0
   SETB   TR1            ;启动定时器 T1
LL:SJMP   LL             ;循环
T_0:MOV   TH0,#15H       ;给 T0 重新赋值
   MOV    TL0,#0A0H      ;
   CPL    P1.2           ;定时到，取反 P1.2
   RETI                  ;
T_1:MOV   TH1,#0FCH      ;给 T1 重新赋值
   MOV    TL1,#18H       ;
   CPL    P1.1           ;计数到，取反 P1.1
```

```
    RETI                    ;
END。
```

把程序下载到单片机，看到什么？LED3 在不断的闪烁，这就是 T0 的作用，闪烁的周期是多少，请大家计算一下。接下来，把 P1.2（也就是 3 脚）和 P3.5（也就是 T1）的输入端相连接，是不是接在 P1.1 上的 LED2 每 2 秒闪烁一次。对了，这就是 T1 作计数器的结果。

在这段程序里，有一点请大家注意☺，第四条—AJMP　T_0,为什么要在 T 和 0 之间加上一条横线，而不直接用 T0 呢？原来在 MCS—51 系列单片机中，是不能用 T0、T1、INT、RET、IP、PSW 等等内部名称作为标号的，如果这样做的话，编译软件会出错，这点我们好象很早以前曾经提到过。接下来我们再来做一个外部中断的实验：

四．外部中断实验

程序如下：

```
ORG   0000H     ;
AJMP   START   ;
ORG   0003H     ;外部中断 0 地址入口
AJMP   INTO     ;
ORG   30H       ;
START: MOV   SP,#5FH   ;
   MOV   P1,#0FFH       ;灯全灭
   MOV   P3,#0FFH       ;P3 口置高电平
   SETB   EA            ;
   SETB   EX0           ;
   AJMP   $             ;
INTO:PUSH   ACC        ;
   PUSH   PSW          ;
   CPL   P1.0          ;
   POP   PSW          ;
   POP   ACC           ;
   RETI                ;
END。
```

本程序的功能就是按一次按键 S1（接在 P3.2 引脚上的），就引发一次外部中断（INT0=0），取反一次 P1.0，因此理论上按一下灯亮，再按一下灯灭，有点象我们工程应用中的自锁开关。不过这段程序在实际的实验中，可能会发觉有时不很“灵”，按了它没反应，但在大部份时候还是对的，这是怎么回事呢？其实这是因为按键没有作“去抖动”处理，也就是说，理论上我们是按了一次键，但由于计算机的处理速度很快，计算机实际上却认为已经按了好多次了，如何来解决这个问题呢？这就需要我们对按键作去抖动处理，什么是按键的去抖动处理，我们下一课讨论键盘接口时再作详细解释。

五．本课总结

通过这两节课的实验，我们对定时/计数器和中断的使用方法已经有了一个基本的了解，希望大家继续多做实验，本来嘛，学会单片机靠的就是不断的实践和总结。

第二十一课 键盘接口与编程（一）

键盘接口和数码管接口是构成单片机人机界面的主要方法，对于一个初学者来说，这部分的内容也是较难的，我们将用四节课的时间来学习这方面的知识。这一课先来讨论键盘的接口原理与编程方法。

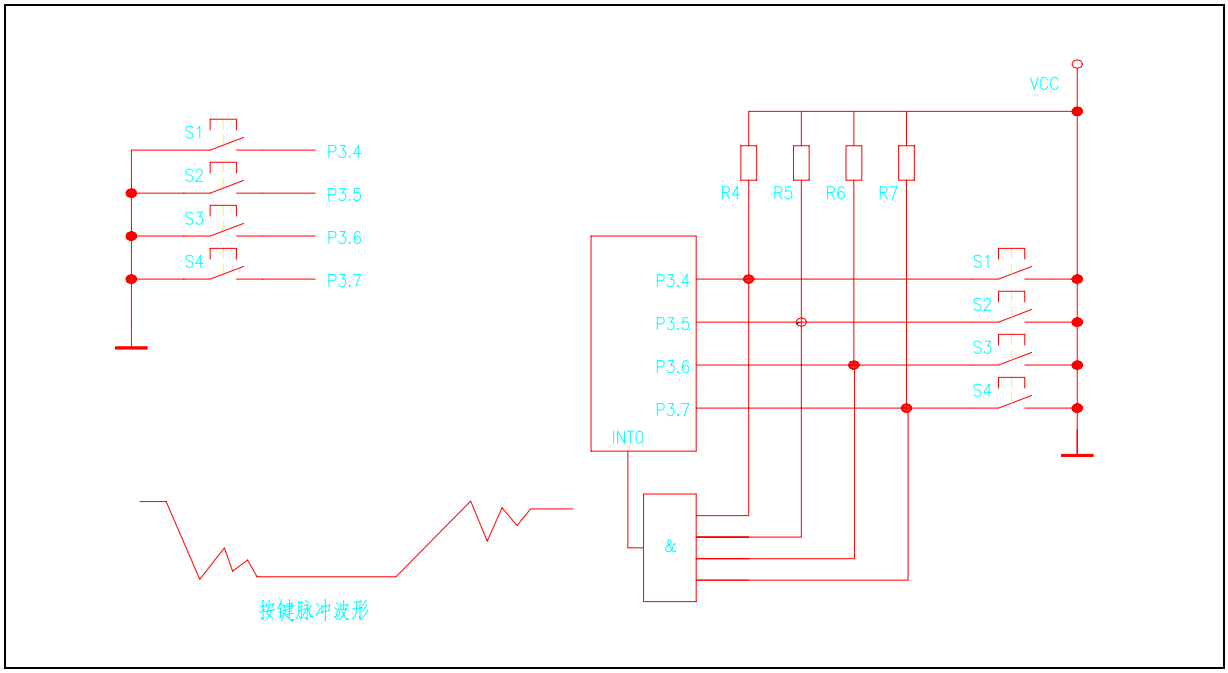
键盘是单片机应用系统不可缺少的重要输入设备，主要负责向计算机传递信息，我们可以通过键盘向计算机输入各种指令、地址和数据。它一般由若干个按键组合成开关矩阵，按照其接线方式的不同可分为两种：一种是独立式接法，一种是矩阵式接法（如下面的图），这一课先来讲解独立式键盘的工作原理和编程方法。

一．独立式键盘的工作原理和编程方法

独立式键盘具有结构简单，使用灵活等特点，因此被广泛应用于单片机系统中，那么它是如何来工作的呢？我们慢慢往下看：

1．独立式键盘的接线原理

独立式键盘是由若干个机械触点开关构成的，把它与单片机的 I/O 口线连起来，通过读 I/O 口的电平状态，即可识别出相应的按键是否被按下，下面的左图就是我们实验板的按键连接图：



如果按键不被按下，其端口就为高电平；如果相应的按键被按下，则端口就变为低电平。在这种键盘的连接方法中，我们通常采用下拉电平接法，即各按键开关一端接低电平，另一端接单片机 I/O 口线，如上左图所示。这是为了保证在按键断开时，各 I/O 口线有确定的高电平。

通常我们用来做键盘的按键有触点式和非触点式两种，单片机中应用的一般是由机械触点构成的触点式微动开关。这种开关具有结构简单，使用可靠的优点，但当我们按下按键或释放按键的时候它有一个特点，就是会产生抖动，看上图的按键脉冲波形，这种抖动对于人来说是感觉不到的，但对单片机来说，则是完全可以感应到的，因为单片机处理的速度是在微秒级的，而机械抖动的时间至少是毫秒级，对计算机而言，这已是一个很“漫长”的过程了。下面我们通过一个实验来验证一下，实验程序如下：

```
ORG   0000H          ;
AJMP  START          ;
ORG   0030H          ;
START:MOV  SP,5FH    ;
```

```
    MOV  P1,#0FFH   ;
    MOV  P3,#0FFH   ;
L1:JNB  P3.4,L2   ;按下按键开关，取反一次 P1.0（灯亮），再按一下灯灭
    JNB  P3.5,L3   ;按下按键开关，取反一次 P1.1（灯亮），再按一下灯灭
    LJMP  L1        ;
L2:CPL  P1.0      ;
    LJMP  L1        ;
L3:CPL  P1.1      ;
    LJMP  L1        ;
END。
```

把这个程序下载到单片机，我们会发现，当按下相应的按键时，灯并不是想象中的按一下亮，再按一下就灭，而是有时灵，有时不灵，为什么会这样呢？原来，当你按了一次按键，可是单片机却早已执行了好多次，如果执行的次数正好是奇数次，那么结果正如你所料；如果执行的次数是偶数次，那结果就不对了。为了使 CPU 能正确地读出端口的状态，对每一次按键只作一次响应，就必须考虑如何去除按键的抖动。

2．按键的去抖动原则和方法

常用的去抖动的方法有两种：硬件方法和软件方法。硬件去抖动的方法很多，好多书都有介绍，这不在我们的讨论范围。单片机中常用软件去抖动法，软件法其实也很简单，就是在单片机获得端口为低电平的信息后，不是立即认定按键已被按下，而是延时 10 毫秒或更长一些时间后再次检测该端口，如果仍为低，说明此键的确被按下了，这实际上是避开了按键按下时的抖动时间；而在检测到按键释放后（端口为高电平时）再延时 5-10 毫秒，消除后沿的抖动，然后再对按键进行处理，不过一般情况下，我们通常不对按键释放的后沿进行处理，实践证明，也能满足通常的要求。下面我们把前面的程序改一下，看看按键的去抖动是如何实现的。看下面的程序：

```
    ORG  0000H        ;
    AJMP  START       ;
    ORG  0030H        ;
    START:MOV  SP,#5FH ;
      MOV  P1,#0FFH   ;
      MOV  P3,#0FFH   ;
    L1:JB  P3.4,L2    ;P3.4 为“1”，不做处理，转 P3.5，否则说明有键按下
      LCALL  D10mS    ; 调用延时程序，去除抖动
      JB  P3.4,L1     ; P3.4 为“0”，说明此键确实被按下了
      CPL  P1.0       ;取反 P1.0
    L3:JNB  P3.4,L3   ;直到 P3.4 释放后转去判断第二个键
    L2:JB  P3.5,L1    ;P3.5 为“1”，返回去继续处理 P3.4，否则说明有键按下
      LCALL  D10mS    ; 调用延时程序，去除抖动
      JB  P3.5,L2     ; P3.5 为“0”，说明此键确实被按下了
      CPL  P1.1       ; 取反 P1.1
    L4:JNB  P3.5,L4   ;直到 P3.5 释放为止
      LJMP  L1        ;返回

    D10mS:MOV  R7,#50 ;延时的时间一般为 5-20mS
      D1:MOV  R6,#100   ;
      D2:DJNZ  R6,D2    ;
      DJNZ  R7,D1     ;
```

```
    RET             ;
END。
```

把这段程序写入单片机，试试看，是不是行了，这就是独立式按键去抖动的基本方法。不过这个程序在实际应用中并没有多大的意义，因为如果按键数量比较多的话，程序就会变得很长，为什么会这样呢？因为这里我们采用了直接寻址的方式，如果我们把键值放入一个表格中，再通过查表程序来判断到底是哪个按键被按下了，再去处理相应的程序就会很简单，想想看，该怎么做？

二．独立式键盘的编程方法

我们刚才的程序演示了按键的去抖动原理和基本方法，接下来让我们做一个按键使用的实验来验证一下，大家看附图的电路图，我们的实验板上有 4 个按键分别接到了 P3 口的 P3.2,P3.3,P3.4,P3.5 引脚上，现在我们用 P3.2,P3.3,P3.4 和 P3.5 这四个按键来做一个实验。实验之前，先定义各个按键的功能：

A．P3.2　开始，按此键则灯开始流动(由左向右)
B．P3.3　停止，按此键则停止流动，所有灯为灭
C．P3.4　向左，按此键则灯反向流动（由右向左）
D．P3.5　向右，按此键则灯正向流动（由左向右）

实验程序如下：

```
UpDown   EQU  00H  ;            上下行标志
StartEnd  EQU  01H;            起动及停止标志
LampCode  EQU  21H;            存放流动的数据代码
ORG  0000H        ;
AJMP  MAIN        ;
ORG  30H          ;
MAIN:MOV  SP,#5FH ;
  MOV  P1,#0FFH     ;
  CLR  UpDown       ;启动时处于向上的状态
  CLR  StartEnd     ;启动时处于停止状态
  MOV  LampCode,#0FEH;单灯流动的代码
  LOOP:ACALL  KEY    ;调用键盘程序
  JNB  F0,LNEXT      ;如果无键按下，则继续
  ACALL  KEYPROC     ;否则调用键盘处理程序
  LNEXT:ACALL  LAMP  ;调用灯显示程序
  AJMP  LOOP         ;反复循环，主程序到此结束
DELAY:MOV  R7,#100   ;
  D1:MOV  R6,#100    ;
  DJNZ  R6,$         ;
  DJNZ  R7,D1        ;
  RET                ;延时程序，键盘处理中调用
KEYPROC:MOV  A,B     ;从 B 寄存器中获取键值
  JB  ACC.2,KeyStart ;分析键的代码，某位被按下，则该位为"1"（在键盘程序中已取反）
  JB  ACC.3,KeyOver  ;
  JB  ACC.4,KeyUp    ;
  JB  ACC.5,KeyDown  ;
  AJMP  KEY_RET      ;
KeyStart:SETB  StartEnd ;第一个键按下后的处理
  AJMP  KEY_RET       ;
```

```
KeyOver:CLR  StartEnd;第二个键按下后的处理
AJMP  KEY_RET       ;
KeyUp:SETB  UpDown  ;第三个键按下后的处理
AJMP  KEY_RET       ;
KeyDown:CLR  UpDown ;第四个键按下后的处理
KEY_RET:RET         ;
KEY:CLR  F0         ;清 F0，表示无键按下
ORL  P3,#01111000B  ;将 P3 口接有四个键的位置“1”
MOV  A,P3           ;取 P3 口的值
ORL  A,#10000111B   ;将其余四位也置“1”
CPL  A              ;取反
JZ  K_RET           ;如果为“0”则无键按下
ACALL  DELAY        ;否则延时去键抖
ORL  P3,#01111000B  ;
MOV  A,P3           ;
ORL  A,#10000111B   ;
CPL  A              ;
JZ  K_RET           ;
MOV  B,A            ;确实有键按下，将键值存入 B 中
SETB  F0            ;设置有键按下的标志
K_RET:ORL  P3,#01111000B ;此处循环等待键的释放
MOV  A,P3           ;
ORL  A,#10000111B   ;
CPL  A              ;
JZ  K_RET1          ;直到读取的数据取反后为“0”说明键释放了，才从键盘处理程序返回
AJMP  K_RET         ;
K_RET1:RET          ;
D500mS:             ;流水灯的延迟时间
PUSH  PSW           ;
SETB  RS0           ;
MOV  R7,#200        ;
D51:MOV  R6,#250    ;
D52:NOP
NOP
NOP
NOP
DJNZ  R6,D52    ;
DJNZ  R7,D51    ;
POP  PSW        ;
RET             ;
LAMP:JB  StartEnd,LampStart ;如果 StartEnd=1，则启动
MOV  P1,#0FFH   ;
AJMP  LAMPRET   ;否则关闭所有显示，返回
LampStart:JB  UpDown,LAMPUP ;如果 UpDown=1，则向上流动
MOV  A,LAMPCODE  ;
```

```
        RL   A              ;实际就是左移位
        MOV  LAMPCODE,A     ;
        MOV  P1,A           ;
        LCALL  D500mS       ;
        AJMP  LAMPRET       ;
        LAMPUP:MOV  A,LAMPCODE ;
        RR   A              ;向下流动实际就是右移
        MOV  LAMPCODE,A     ;
        MOV  P1,A           ;
        LCALL  D500mS       ;
        LAMPRET:RET         ;
        END。
```

这段程序是我们到目前为止最长的程序，相信大多数指令大家应该能看懂，开始三条，UpDown EQU 00H；StartEnd EQU 01H；LampCode EQU 21H 给大家解释一下，EQU 叫做等值伪指令，它的功能是将一个常数或者特定的符号赋予规定的字符串。什么意思呢？举个例子：

```
ORG  200H;
ABC  EQU  R6;
MOV  A，ABC;
```

这里将 ABC 等值为寄存器 R6，也就是说，在指令中，R6 这个寄存器可以用字符串 ABC 来代替，为什么要这样写呢？当然是为了增加程序的可读性，不过有一点大家要记住了，这里使用的字符串不是标号，不能用“：”来做分隔符，比如这样写 ABC:EQU R6；如果加上“：”汇编程序会出错；当然，用 EQU 指令除了可以赋值数据和地址外，还可以赋值直接地址或者直接当作一个立即数来使用，例如：

```
ABC  EQU  10H;
DELAY  EQU  05AFH;
MOV  A，ABC;
LCALL  DELAY;
```

这里 ABC 赋值以后被当作了直接地址使用；而 DELAY 被赋值以后则成了一个 16 位的地址。

如此一来，上面的三条指令也就很清楚了。这里有一个问题大家需注意☺：使用 EQU 伪指令必须先赋值，后使用，所以一般的程序都把赋值指令放在程序的开头部分。

既然讲到了赋值伪指令，我们再讲一下另外三条赋值伪指令.

A．位地址定义伪指令 BIT

它的功能是将一个可直接寻址的位地址赋予所规定的字符名称，例如：

ABC BIT P1.0；把 P1.0 赋值给 ABC，即字符串 ABC 就是直接寻址位 P1.0。

这里注意☺：与 EQU 不同的是，这条指令只能对位地址赋值，而不能对寄存器或直接地址和立即数赋值。相反，EQU 指令却可以用来定义位地址变量，不过这时所赋的值应当是具体的位地址值。比如 P1.0 要用 90H 来代替；P2.0 要用 AOH 来代替等等。

B．内部 RAM 定义伪指令 DATA

它的功能是给一个 8 位的内部 RAM 起一个名称，例如：

ABC DATA 20H；把内部 RAM 的 20H 定义为 ABC。

C．外部 RAM 定义伪指令 XDADT

给一个 8 位的外部 RAM 起一个名称，例如：

ABC XDATA 0ACH；由于 89C51 的内部 RAM 寻址范围为 00H-FFH，所以这个地址必然大于 FFH。

讲了赋值伪指令，再回到上面的按键处理程序，这段程序的功能虽然很简单，但它演示了一个键盘处理程序的基本思路，程序本身很简单，也不很实用，实际工作中还会有好多要考虑的因素，比如主循环每次都调用了灯的循环程序，会造成按键反应“迟钝”；而如果一直按着键不放，则灯不会再流动，

一直要到松开手为止，大家可以仔细考虑一下这些问题，想想有什么好的解决办法。

独立式键盘除了上面介绍的这种连接方法，我们还可以采用上图右边所示的连接方法，用一个“与非”门把四个输入端连接起来，当有任何一个按键按下时，都会使“与非”门输出为低电平，从而引起单片机的中断，它的好处是不用在主程序中不断地循环查询了，如果有键按下，单片机就去作相应的处理。具体的应用就不介绍了，大家如果有兴趣的话可以自己找一些相关的资料看一下。

三．本课总结

这一课通过两个实验讲解了独立式键盘的工作原理和基本的去抖动方法，由于键盘的实际使用是千差万别的，所以工程中您还得根据实际情况灵活应用，这里只能给大家一个基本的认识。

四．第 21 课习题

1．什么是键盘的去抖动问题？为什么要对键盘进行去抖动处理？

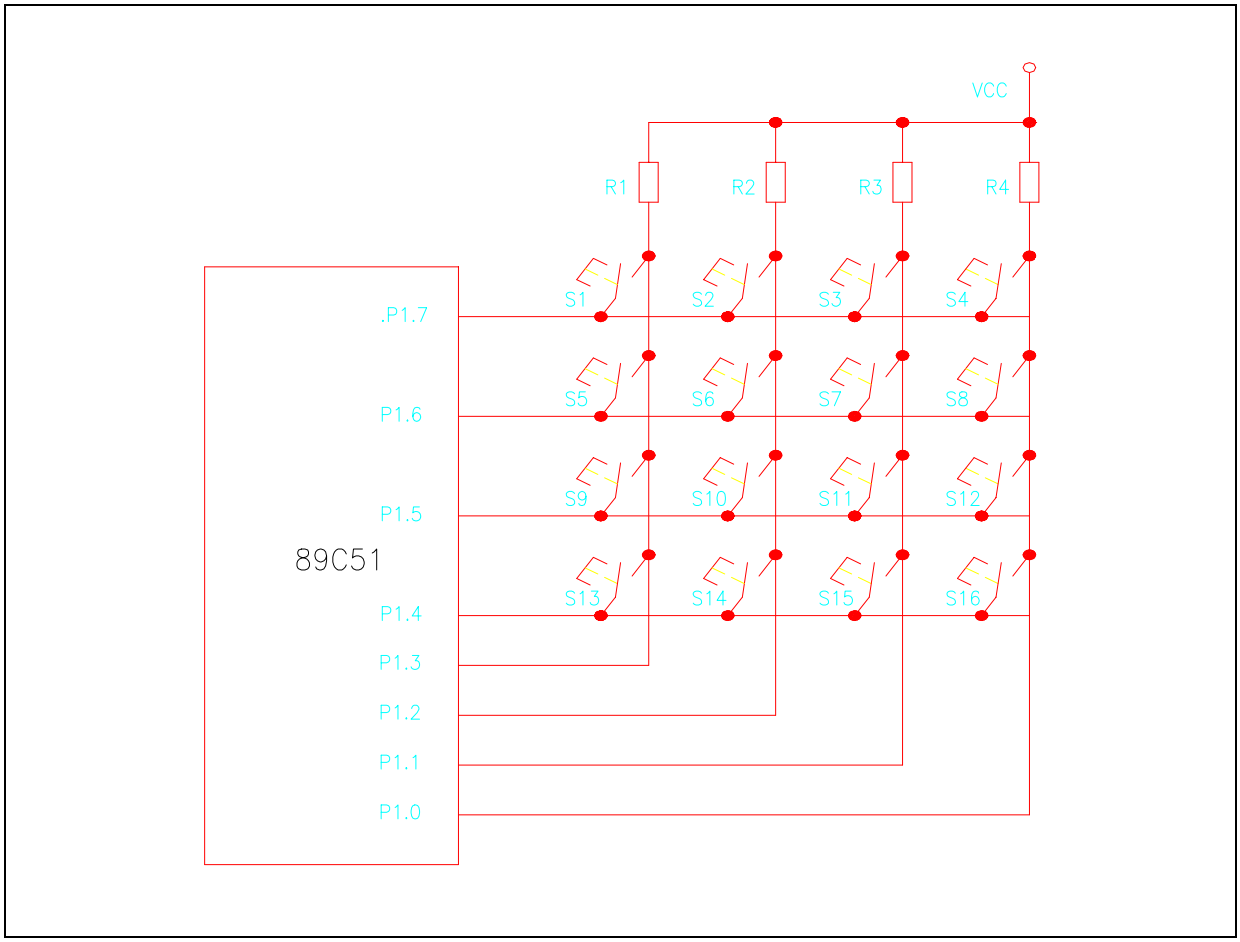
2．找一个硬件的去抖动电路，并自行分析其工作原理。

第二十二课 键盘接口与编程（二）

上一课我们讨论了独立式键盘的工作原理和编程方法，这一课再来讲解矩阵式键盘的工作原理和编程方法。

一． 矩阵式键盘的连接方法和工作原理：

什么是矩阵式键盘？当键盘中按键数量较多时，为了减少 I/O 口线的占用，通常将按键排列成矩阵形式，如下图所示。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样做有什么好处呢？大家看下面的电路图，一个并行口可以构成 4*4=16 个按键，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别就越明显。比如再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一个键（9 键）。由此可见，在需要的按键数量比较多时，采用矩阵法来连接键盘是非常合理的。



矩阵式结构的键盘显然比独立式键盘复杂一些，识别也要复杂一些，在上图中，列线通过电阻接电源，并将行线所接的单片机 4 个 I/O 口作为输出端，而列线所接的 I/O 口则作为输入端。这样，当按键没有被按下时，所有的输出端都是高电平，代表无键按下，行线输出是低电平；一旦有键按下，则输入线就会被拉低，这样，通过读入输入线的状态就可得知是否有键按下了，具体的识别及编程方法如下所述：

二．矩阵式键盘的按键识别方法

确定矩阵式键盘上任何一个键被按下通常采用“行扫描法”或者“行反转法”。行扫描法又称为逐行（或列）扫描查询法，它是一种最常用的多按键识别方法。因此我们就以“行扫描法”为例介绍矩阵式键盘的工作原理：

1. 判断键盘中有无键按下

将全部行线 X0-X3 置低电平，然后检测列线的状态，只要有一列的电平为低，则表示键盘中有键被按下，而且闭合的键位于低电平线与 4 根行线相交叉的 4 个按键之中；若所有列线均为高电平，则表示键盘中无键按下。

2. 判断闭合键所在的位置

在确认有键按下后，即可进入确定具体闭合键的过程。其方法是：依次将行线置为低电平（即在置某根行线为低电平时，其它线为高电平），当确定某根行线为低电平后，再逐行检测各列线的电平状态，若某列为低，则该列线与置为低电平的行线交叉处的按键就是闭合的按键。

下面给出一个具体的例子：

单片机的 P1 口用作键盘 I/O 口，键盘的列线接到 P1 口的低 4 位，键盘的行线接到 P1 口的高 4 位，也就是把列线 P1.0-P1.3 分别接 4 个上拉电阻到电源，把行线 P1.4-P1.7 设置为输出线，4 根行线和 4 根列线形成 16 个相交点，如上图所示。

检测当前是否有键被按下：检测的方法是 P1.4-P1.7 输出全“0”，读取 P1.0-P1.3 的状态，若 P1.0-P1.3 为全“1”，则说明无键闭合；否则有键闭合。

去除键抖动：当检测到有键按下后，延时一段时间再做下一次的检测判断，若仍有键按下，应识别出是哪一个键闭合，方法是对键盘的行线进行扫描，P1.4-P1.7 按下述 4 种组合依次输出：P1.7-1110；P1.6-1101；P1.5-1011；P1.4-0111；在每组行输出时读取 P1.0-P1.3；若全为“1”，则表示为“0”一行的没有键闭合；否则就是有键闭合。由此得到闭合键的行值和列值，然后可采用计算法或查表法将闭合键的行值和列值转换成所定义的键值。为了保证按键每闭合一次 CPU 仅作一次处理，必须去除键释放时的抖动。看下面的实例：

三. 矩阵式键盘的实验程序

```
      ORG   0030H          ;
      SCAN: MOV   P1,#0FH  ;
        MOV   A,P1         ;
        ANL   A,#0FH       ;
        CJNE  A,#0FH,NEXT1 ;
        SJMP  NEXT3        ;
        NEXT1:ACALL  D20Ms ;
        MOV   A,#0EFH      ;
        NEXT2:MOV  R1,A    ;
        MOV   P1,A         ;
        MOV   A,P1         ;
        ANL   A,#0FH       ;
        CJNE  A,#0FH,KCODE ;
        MOV   A,R1         ;
        SETB  C            ;
        RLC   A            ;
        JC   NEXT2         ;
      NEXT3:MOV  R0,#00H   ;
        RET                ;
      KCODE:MOV  B,#0FBH   ;
      NEXT4:RRC  A         ;
        INC   B            ;
        JC   NEXT4         ;
        MOV   A,R1         ;
```

```
    SWAP   A            ;
NEXT5:RRC   A           ;
    INC   B             ;
    JC   NEXT5          ;
NEXT6:MOV   A,P1        ;
    ANL   A,#0FH        ;
    CJNE   A,#0FH,NEXT6;
    MOV   R0,#0FFH      ;
    RET                 ;
END。
```

由于我们的实验设备不能做这个实验，所以上面的程序只能请大家自行分析一下，事实上开发多键盘的应用系统是非常复杂的，为了不增加大家的学习难度，这里只给大家讲一个基本的概念，您只要稍微了解一下就可以了，详细的内容我们将留到下册中再来仔细地讨论。如果你确实在实际的应用中碰到了这样的问题，可以电话或者 E-mail 与我联系，我会尽力的帮助你大家。

四．本课总结

事实上，在比较复杂的单片机系统中，键盘的的设计只是程序设计的一部分，在这种系统中，还会有很多的其他程序，设计这种系统也是一项非常复杂的工作，作为初学者我们对此有所了解就已经可以了，若您想尽快的掌握这方面的知识，请继续学习教程的下册部分。

五．第 22 课习题

1．矩阵式键盘采用什么样的扫描方式？

2．识别矩阵式键盘包括哪几个步骤？

第二十三课 数码管接口与编程（一）

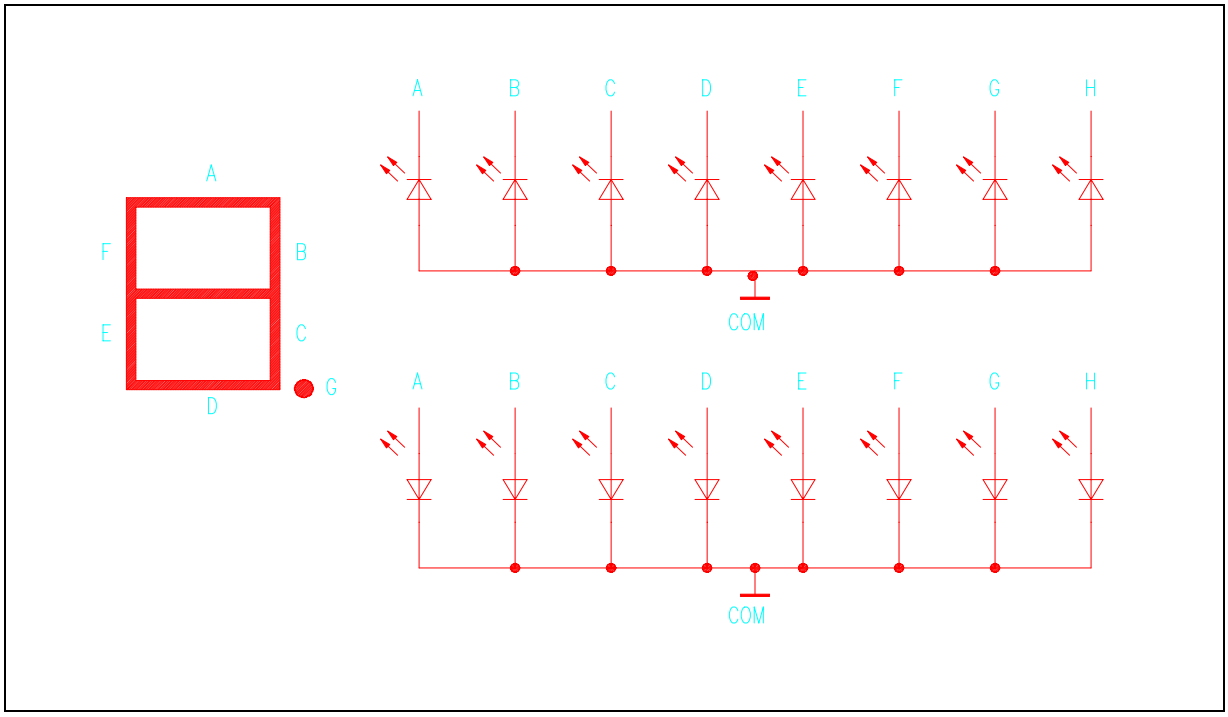
如果说键盘构成的是单片机的输入系统，那么数码管就是单片机的输出系统，和学会使用键盘接口一样，学会数码管的接口与编程对单片机的开发同样有着十分重要的意义。这一课我们就来讲解数码管的接口与编程方法：

一．LED 数码显示器的连接与显示方法

在单片机系统中，通常用 LED 数码显示器来显示各种数字或符号，由于它具有显示清晰、亮度高、使用电压低、寿命长的特点，因此使用非常广泛。那么数码管是如何工作的呢？我们学习数字电路时讲过，可能大家都忘得差不多了，这里我再从头给大家讲一遍，您可不要觉得我烦哦！

还记得我们小时候玩过的“火柴棒游戏”吗，几根火柴棒组合起来，可以拼成各种各样的图形，LED 显示器实际上就是利用这个原理做成的。

八段 LED 显示器由 8 个发光二极管组成。其中 7 个长条形的发光管排列成一个“日”字形，另一个圆点形的发光管在显示器的右下角作为显示小数点用，它能显示各种数字及部份英文字母。LED 显示器有两种不同的连接形式：一种是 8 个发光二极管的正极连在一起，称之为共阳极 LED 显示器；另一种是 8 个发光二极管的负极连在一起，称之为共阴极 LED 显示器。它们的内部电路图如下所示：

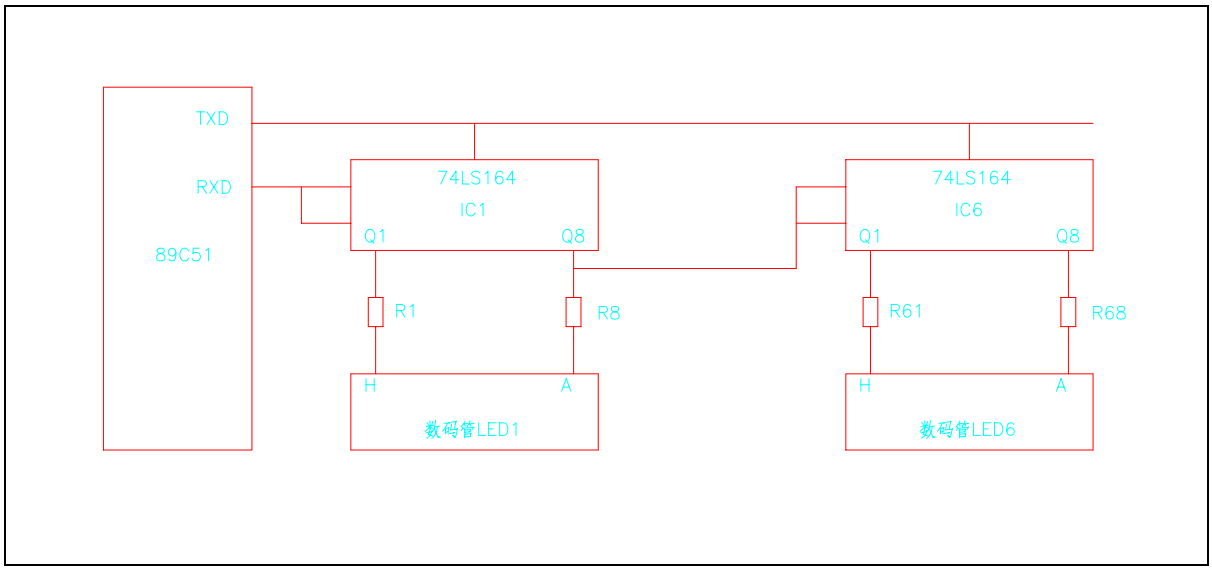


（勘误：图中有一点点小错误，把 H 端漏了，由于这是用 CAD14 画的，现在改起来不方便，请大家注意一下）由图可以看出，共阳和共阴结构的 LED 显示器各笔划段名的安排位置是相同的，当二极管导通时，相应的笔划段就发亮，由发亮的笔划段组合而显示出各种字符。八个笔划段 h（在许多书中用 dp 来表示，其实是一个意思）gfedcba 对应于一个字节（8 位）的 D7 D6 D5 D4 D3 D2 D1 D0,于是用 8 位二进制码就可以表示欲显示字符的字形代码。例如，对于共阴 LED 显示器，当公共阴极接地（零电平），阳极 hgfedcba 各段为 01110011 时，显示器就显示“P”字符，即“P”字符的字形码是 73H；而如果是共阳极 LED 显示器，公共阳极接高电平，显示“P”字符的字形代码应为 10001100（8CH）。这里必须注意☺的是：很多产品为了方便接线，常常不按照规则的方法去对应字段与位的关系，这时字形码就必须根据接线来自行设计了。后面我们会给出一个例子让大家参考，那么数码管和单片机又是如何连接的呢？请继续往下看：

二．LED 数码管的静态显示方法

在单片机的应用系统中，数码管显示器的显示常采用两种方法：静态显示和动态扫描显示。所谓静态显示，就是把多个 LED 显示器的每一段与一个独立的并行口连接起来，而公共端则根据数码管的种类连接到“VCC”或“GND”端，这种连接方式的每一个显示器都要占用一个单独的具有锁存功能的 I/O 端口，用于笔划段字形代码，单片机只需把要显示的字形代码发送到接口电路，就不用再管它了，直到要显示新的数据时，再发送新的字形码。因此，使用这种方法当显示位数较多时单片机中 I/O 口的开销很大，需要提供的 I/O 接口电路也较复杂，但它具有编程简单，显示稳定，CPU 的效率较高的优点。

下面我们以常用的串-并转换电路 74LS164 为例，介绍一种常用的静态显示电路，以使大家对静态显示有一个基本的了解。请看下面的电路图：



MCS-51 单片机串行口方式称为移位寄存器方式，外接 6 片 74LS164 作为 6 位 LED 显示器的静态显示接口，我们把单片机的 RXD 作为数据输出线，TXD 作为移位时钟脉冲。74LS164 为 TTL 单向 8 位移位寄存器，可实现串行输入，并行输出。其中 A、B（1、2 脚）为串行数据输入端，2 个引脚按逻辑“与”运算规律输入信号，只有一个输入信号时可并接。T（8 脚）为时钟输入端，可连接到串行口的 TXD 端。每一个时钟信号的上升沿加到 T 端时，移位寄存器移一位，8 个时钟脉冲过后，8 位二进制数全部移入 74LS164 中；R（9 脚）为复位端，当 R=0 时，移位寄存器各位复“0”，只有当 R=1 时，时钟脉冲才起作用。Q1……Q8（3-6 脚和 10-13 脚）为并行输出端，分别接到 LED 显示器的 hgfedcba 各段对应的引脚上。

关于 74LS164 还可以作如下的介绍：所谓时钟脉冲端，其实就是需要高、低、高、低的脉冲，不管这个脉冲是怎么来的，比如，我们用根电线，一端接 T，一端用手拿着，分别接高电平、低电平，那也是给出时钟脉冲，在 74LS164 获得时钟脉冲的瞬间（再讲清楚点，是在脉冲的前沿），如果数据输入端（1，2 脚）是高电平，则就会有一个“1”进入到 74LS164 的内部；如果数据输入端是低电平，则就会有一个“0”进入其内部。在给出了 8 个脉冲后，最先进入 74LS164 的第一个数据到达了最高位，然后再来一个脉冲会有什么情况发生呢？第一个脉冲就会从最高位移出，就象车站排队买票，栏杆就那么长，要从后面进去一个人，就必须要从前面走出去一个人才行。

搞清了这一点，让我们再来看电路图（电路图在后面的附录中），6 片 74LS164 首尾相串，而时钟端则连接在一起，这样，当第一次输入 8 个脉冲时，从单片机 RXD 端输出的数据就进入到了第一片 74LS164 中了；而当第二次 8 个脉冲到来后，这个数据就进入到了第二片 74LS164，新的数据则进入了第一片 74LS164 中……这样，当第六次 8 个脉冲完成后，首次送出的数据被送到了最左面的 74LS164 中，其他的数据则依次出现在第一、二、三、四、五片 74LS164 中。这里有个问题，在第一次 8 个脉冲到来时，除了第一片 74LS164 中接收数据外，其他各片在干吗呢？它们也在接收数据，因为它们的时钟端都是被接在一起的，可是数据还没有送到其他各片呢，它们在接收什么数据呢？其实所谓数据不过是一种说法而已，它实际上就是电平的高或低。当第一次 8 个脉冲到来时，第一片 74LS164 固然是从单片机接收数

据了，而其它各片也接到前一片的 Q8 上，而 Q8 是一根电线，在数字电路中它只可能有两种状态：低电平或高电平，也就是“0”和“1”。所以它的下一片 74LS164 也相当于是在接收数据，只是接收的全部是“0”或“1”而已。这个问题放在这儿来讲，可能有的朋友不屑一顾，而有的朋友可能还是不清楚，这实际上涉及到数的本质的问题，如果不懂，请并回到前面的内容再仔细的复习一遍，或找一些数字电路方面的书籍看一下。理解了 74LS164 的工作原理，再来看这个问题就会变得简单多了，这里就算给大家留个习题吧，希望大家务必把 74LS164 的工作原理搞清楚，搞懂了这一点，您的级别就超过初级了，变成中级了。

接下来让我们做一个实验，做这个实验也需要一套附件，这套附件由于大家只用一次，所以各位可以到我这儿来借，同样免费使用，不过还是那句老话，请各位多多爱惜哦，不要把它搞坏了。

我们把附件的两根线连接到实验板的 P3 口扩展插座和实验仪的电源接口上，先看实验要求：

输入：把要显示的数分别放在显示缓冲区 60H-65H 共 6 个单元中，并且分别对应各个数码管 LED1-LED6。

输出：将预置在显示缓冲区中的 6 个数转换成相应的显示字形码，然后输出到显示器中显示出来。程序如下：

```
ORG   0000H          ;
AJMP   START         ;
ORG   30H            ;
START:MOV   SP,#6FH;
MOV   65H,#0         ;
MOV   64H,#1         ;
MOV   63H,#2         ;
MOV   62H,#3         ;
MOV   61H,#4         ;
MOV   60H,#5         ;
LCALL   DISP         ;
SJMP   $             ;
DISP:MOV   SCON,#00H    ;初始化串行口方式 0
  MOV   R1,#06H         ;显示 6 位数
  MOV   R0,#65H         ;60H-65H 为显示缓冲区
  MOV   DPTR,#SETTAB    ;字形表的入口地址
  LOOP:MOV   A,@R0      ;取最高位的待显示数据
  MOVC   A,@A+DPTR      ;查表获取字形码
  MOV   SBUF,A          ;送串口显示
  DELAY:JNB   TI,DELAY ;等待发送完毕
  CLR   TI              ;清发送标志
  DEC   R0              ;指针下移一位，准备取下一个待显示数
  DJNZ   R1,LOOP        ;直到 6 个数据全显示完
  RET                   ;
SETTAB:                 ;字形表，前面有介绍，后面我们会介绍字形表的制作
DB 03H,9FH,27H,0DH,99H,49H,41H,1FH,01H,09H,0FFH;
END。
```

如果按图示数码管排列，则以上程序将显示 543210。

不过我们的实验板是做不了这个实验的，为什么，我们稍候再讲，先让我们来讲解一下字形表的制作问题，先就上述标准的图形来做，写出数据位和字形的对应关系并列一个表如下（共阳接法，也就是输出为“0”时笔段亮）。

怎么样，不算复杂吧，就是这样列个表格，根据要求（“0”亮“1”灭）写出相应位的“0”和“1”就可以了。接着练习一下，写出显示 A-F 的字形码。

数据位	D7	D6	D5	D4	D3	D2	D1	D0	字型码
笔段位	A	B	C	D	E	F	G	H	
0	0	0	0	0	0	0	1	1	03H
1	1	0	0	1	1	1	1	1	9FH
2	0	0	1	0	0	1	1	1	27H
3	0	0	0	0	1	1	0	1	0DH
4	1	0	0	1	1	0	0	1	99H
5	0	1	0	0	1	0	0	1	49H
6	0	1	0	0	0	0	0	1	41H
7	0	0	0	1	1	1	1	1	1FH
8	0	0	0	0	0	0	0	1	01H
9	0	0	0	0	1	0	0	1	09H

不过这是按照上面的标准接线排列的，在实际的程序设计中，有时为了接线方便常常会把接线顺序打乱（我们的实验板就是这样的），那么此时的字形表又该如何做呢？其实也很简单，一样地列表，以我们的实验板为例，同样显示 9876543210，共阴极接法（注意：和共阳极接法的区别）。接线如下：

P2.0 P2.1 P2.2 P2.3 P2.4 P2.5 P2.6 P2.7 对应 g f a b h c d e，则字形码如下所示：

（0）11101110/EEH；（1）00101000/28H；（2）11001101/ CDH；（3）01101101/ 6DH；（4）00101011/ 2BH；（5）01100111/67H；（6）11100111/E7H；（7）00101100/2CH；（8）11101111/EFH；（9）00101111/ 2FH；

继续练习，写出显示 A-F 的字形码。下面提个问题：如果是共阳极的接法，字形码又该是怎么样的呢？不用我再说了吧，如果学到现在连这个还不明白，那真的是惨了!!!

现在让我们来继续上面的实验，把 543210 的字形码放入上面的查表程序中（即 DB:。。。。。。）后面，如果要显示 012345，我们的程序又该如何修改呢？自己想一下。

三．本课总结

这一课主要讨论数码管的静态显示原理及编程方法。在实际的应用中远比这些要复杂的多，不过，在我们的上册课程中，目的是为了让大家尽快的进入到单片机的应用中来，所以我这里只能讲这么多，如果您想学习更多的知识，只有继续学习课程的下册部分。当然如果您现在就要用到这方面的内容，我可以协助您开发和设计。

四．第 23 课习题

1．什么是 LED 数码管的共阳接法？它与共阴接法有什么不同？

2．深入了解 74LS164 的工作原理。

3．了解 74LS165 的作用和功能。

第二十四课 数码管接口与编程（二）

上一课讲解了数码管的静态显示方法，这一课专门来讨论数码管的动态显示方法：

一．LED 数码显示器的动态显示方法

由于静态显示占用的 I/O 口线较多，CPU 的开销很大，所以为了节省单片机的 I/O 口线，常采用动态扫描方式来作为 LED 数码管的接口电路。在实际的工程应用中，它是使用最为广泛的一种显示方式，其接口电路是把所有显示器的 8 个笔划段 h-a 同名端连在一起，而每一个显示器的公共极 COM 端与各自独立的 I/O 口连接。当 CPU 向字段输出口送出字形码时，所有显示器接收到相同的字形码，但究竟是那个显示器亮，则取决于 COM 端，而这一端是由另外的 I/O 口控制的，所以我们就可以自行决定何时显示哪一位了。而所谓动态扫描就是指我们采用分时的方法，一位一位地轮流控制各个显示器的 COM 端，使各个显示器每隔一段时间点亮一次。

在轮流点亮的扫描过程中，每位显示器的点亮时间是极为短暂的（约 1ms 左右），由于人的视觉暂留现象及发光二极管的余辉效应，尽管实际上各位显示器并非同时点亮，但只要扫描的速度足够快，给人的印象就是一组稳定的显示数据，不会有闪烁感。

在我们实验板的电路图中，我们把 89C52 的 P2 口作为位选端（即同名端 abcdefgh）并联起来，而把它们的片选端分别与 P3.5 和 P3.6 连接(图中为了增加 P3.5 和 P3.6 的驱动能力采用了一个三极管）这样由 P3.5 和 P3.6 控制对应数码管的亮或灭，只要给 P2.0 送入不同的字形码，就能显示不同的数了。下面的这个程序，就是用我们实验板上的两个数码管来显示“0”和“1”，电路的连接方法请看后面的实验板电路图。

```
FIRST   EQU   P3.6          ;第一位数码管的位控制
SECOND  EQU   P3.5          ;第二位数码管的位控制
DISPBUFF  EQU  5AH          ;显示缓冲区为 5AH
ORG   0000H                 ;
AJMP   START                ;
ORG   30H                   ;
START:MOV   SP,#5FH         ;设置堆栈
  MOV   P0,#0FFH            ;
  MOV   P1,#0FFH            ;
  MOV   P2,#0FFH            ;初始化所有显示器，LED 灭
  MOV   DISPBUFF,#0         ;第一位显示 0
  MOV   DISPBUFF+1,#1       ;第二位显示 1
LOOP:LCALL   DISP           ;调用显示程序
  AJMP LOOP                 ;主程序到此结束

DISP:PUSH   ACC             ;ACC 入栈
  PUSH   PSW                ;PSW 入栈
  MOV   A,DISPBUFF          ;取第一个待显示数
  MOV   DPTR,#DISPTAB       ;字形表首地址
  MOVC   A,@A+DPTR          ;取数字 0 的字形码
  MOV   P2,A                ;将字形码送 P2 口
  SETB   FIRST               ;打开第一位显示器位选端
  LCALL   DELAY             ;延时 1 毫秒
  CLR   FIRST               ;关闭第一位显示器（开始准备第二位的数据）
```

```
        MOV   A,DISPBUFF+1        ;取显示缓冲区的第二位
        MOV   DPTR,#DISPTAB       ;第二个字形码地址
        MOVC   A,@A+DPTR          ;去数字 1 的字形码
        MOV   P2,A                ;将第二个字形码送 P2 口
        SETB   SECOND              ;打开第二位显示器位选端
        LCALL   DELAY             ;延时 1 毫秒
        CLR   SECOND             ;关闭第二位显示器
        POP   PSW                 ;
        POP   ACC                 ;
        RET                       ;
      DELAY:PUSH   PSW            ;
        SETB   RS0                ;
        MOV   R7,#50              ;
        D1:MOV   R6,#10           ;
        D2:DJNZ   R6,$            ;
        DJNZ   R7,D1              ;
        POP   PSW                 ;
        RET                       ;
      DISPTAB:DB 0EEH，28H，0CDH，6DH，2BH，67H，0E7H，2CH，0EFH，2FH；
      END。
```

从上面的例子中可以看出，动态扫描显示必须由 CPU 不断地调用显示程序，才能保证持续不断的显示。上面的这个程序虽然可以实现数字的显示，但过于简单了，我们学到现在也应该做点复杂一些的实验，比如让两个 LED 从 0 显示到 99，不断的循环，这可是很经典的程序，几乎所有用数码管显示的系统都会把它作为初始化的显示程序，看下面的实验程序：

```
a_bit equ 20h          ;个位数存放处
b_bit equ 21h          ;十位数存放处
temp equ 22h           ;计数器寄存器
star:    mov temp,#0   ;初始化计数器
stlop:   acall display
         inc temp
         mov a,temp
         cjne a,#100,next ;=100 重来
         mov temp,#0
next:    ljmp stlop
;显示子程序
display: mov a,temp   ;将 temp 中的十六进制数转换成 10 进制
          mov b,#10    ;10 进制/10=10 进制
          div ab
          mov b_bit,a ;十位在 a
          mov a_bit,b ;个位在 b
          mov dptr,#numtab ;指定查表启始地址
          mov r0,#4
dpl1:     mov r1,#250 ;显示 1000 次
dplop:    mov a,a_bit ;取个位数
          MOVC A,@A+DPTR ;查个位数的 7 段代码
```

```
        mov p2,a ;送出个位的 7 段代码
        clr p0.7 ;开个位显示
        acall d1ms ;显示 1ms
        setb p0.7
        mov a,b_bit ;取十位数
        MOVC A,@A+DPTR ;查十位数的 7 段代码
        mov p2,a ;送出十位的 7 段代码
        clr p0.6 ;开十位显示
        acall d1ms ;显示 1ms
        setb p0.6
        djnz r1,dplop ;100 次没完循环
        djnz r0,dpl1 ;4 个 100 次没完循环
        ret
;************************1MS 延时(按 12MHZ 算)************************
D1MS: MOV  R7,#50      ;
       D1:MOV  R6,#200 ;
       D2:DJNZ  R6,$   ;
       DJNZ  R7,D1
      RET
numtab: db DB 0EEH，28H，0CDH，6DH，2BH，67H，0E7H，2CH，0EFH，2FH；
END
```

细心的朋友可能看出来了，这段指令中有的是用大写字母来写的，而有的是用小写字母来写的，有什么区别吗？在 KEIL C51 中用汇编写程序的时候是不区分大小写的，你完全可以根据自己的书写习惯来写，不过标号是不能前面的用大写，后面的用小写，这样编译时会出错，好了，不多说了，你还是自己试试吧！

这段程序可以完成从 0-99 的显示循环，当然在实际的产品中，不可能只显示两个数字不做其他的事，不过程序的结构和思路是没有问题的。

从上面的程序中我们可以看出，与静态扫描相比，动态扫描的程序稍微有点复杂。但这是值得的，因为它可以大大节省单片机的 I/O 口线资源，所以在实际的工程应用中几乎都采用动态扫描的方法来进行数码管的显示，我们的这个程序也具有一定的通用性，只要改变端口的值及计数器的值就可以显示更多的位数了。

不过正如我前面所说的那样，单片机的程序设计在实际应用中还会考虑很多的其他问题，所以这一课的内容同样只是给大家以后的学习打一个基础。倒是下面的 LED 数码管的选择原则和驱动方法大家有必要了解一下。

二．LED 数码管的选择和驱动

LED 数码管是单片机人机界面输出中用的最多也是最简单的显示方式，由于单片机口线的驱动能力是有限的，所以如何来选择和驱动 LED 数码管是单片机初学者的基本功，下面就来给大家介绍一下。

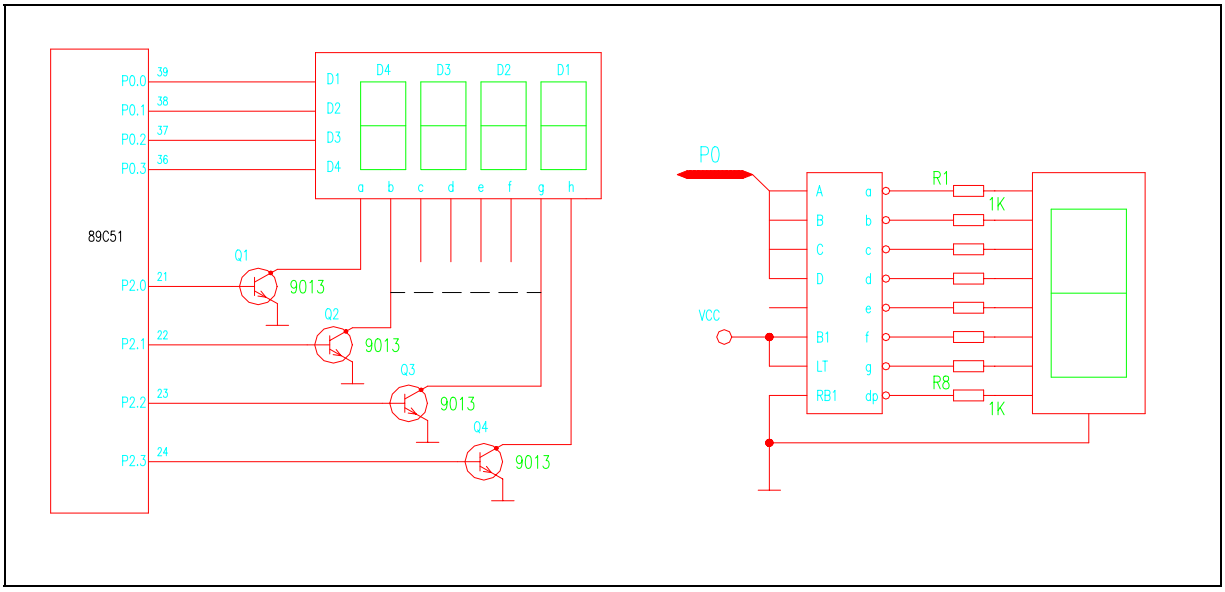
前面我们已经讲过，LED 数码管有两种连接方法：一种是共阳接法；一种是共阴接法。在单片机的应用中，对于共阳接法，我们一般把它叫做“接电源”方法，即把 COM 端接“VCC”，通过控制 COM 端引脚电平的高低来达到片选的目的；而对于共阴接法，则通常叫做“接地”方法，即把 COM 端接“GND”。

由于受单片机口线驱动能力的限制，采用直接驱动的方法，只能连接小规格的 LED 数码管，目前市场上有一种高亮度的数码管，每段工作电流约为 2-3mA，这样当 LED 全亮时，工作电流在 10-20mA 左右，是普通数码管的 1/5，正好能用单片机的口线直接驱动，因此在条件允许的情况下，应尽量采用这种 LED 数码管作为显示器件。

当然如果想用更高亮度或更大尺寸的数码管来作为显示器件，比如户外的电子钟，大型广告牌等

等，就必须采用适当的扩展电路来实现与单片机的接口，常用的接口元件可以是三极管、集成电路和专用芯片等。

三极管的规格可以根据数码管所需的驱动电流大小进行选择，电流比较小的可以用 9013，8550 等小功率晶体管，电流比较大的则可以用 TIP217 等大功率三极管；而当显示器的位数较多时，一般也会采用集成电路来作接口，此类集成电路有 2003、2803、7406、75452 等，它们的功能其实就是由多路晶体管组成的达林顿电路具体电路请大家自己找一些相关的资料查一下；另外还可以使用一种叫译码/驱动器的芯片，这种芯片能将二-十进制码（BCD 码）译成七段码（a-g）以驱动数码管，采用这种芯片的最大好处是编程简单，并且能提高 CPU 的运行效率，如 CD4511 或 74LS47 等就是此类芯片（不过，它们的驱动能力也是有限的，具体数据请参考有关的 DS 介绍）。



（勘误：真是不好意思，左面的这个图又画错了，四个三极管应该用来驱动片选端，而不是位选端，请大家千万别照抄，不然出了问题可不要来找我哦！*^*）

近几年来，国内外厂商还开发了许多基于串行总线（SPI）和 I2C 总线方式的 LED 接口芯片，这些芯片采用 SPI 总线或 I2C 总线方式与单片机进行通讯，具用占用单片机口线少，程序易于实现的特点，比如美信的 MAX7219、力源的 PS7219（SPI 总线）和 SAA1064（I2C 总线）等，有些芯片还集成了键盘控制器，可以实现键盘和显示的双重功能，如 zlg7289 等。关于这方面的内容请您自己找一些相关的资料来看一下。

其实，除了数码管显示外，在实际的工程应用中，单片机还有很多的显示方法，其中比较常用的就是液晶显示器。液晶显示器是一种低压低功耗的显示器件，它的工作电压一般只要几伏，工作电流仅有几个微安，是任何数码管显示器件所无法比拟的。除此之外，液晶显示器的最大优点就是可以显示文字、图形和曲线，与传统的数码管显示器相比，显示界面有了质的提高，采用点阵式的液晶显示器配合大规模集成电路能够显示大量的信息，目前已经广泛使用在各类中高档仪器仪表及家用电器中，比如数字万用表，手机，数码相机等等。不过它的使用方法与编程就比较复杂了，我们只有等到下册中再来给大家讲解了。

二．本课总结

到本课为止，我们课程的上册部分已经全部讲完，希望大家课后多进行练习，因为学习单片机是一种非常注重实践的课程，如果离开了实际的应用，那么学习再多的理论知识也是非常无效的，课程的下册部分我还在继

续的编写之中，等全部写完了会通知大家。

由于本人的才疏学浅，教材中难免有疏漏，甚至有许多不正确的地方，希望大家多多批评指正!!!

最后申明：本教程的编写其实多数参考了平凡老师的教程，只不过看到他写的内容有很多的错误之处，我就进行了大量的整理和修改，并应用到实际的教学工作中。据说他现在的教程已经出版了，如果大家打印不方便，看电子书觉得累，还是去买一本，不要老想着图小便宜，毕竟人家也是化了很多心血的，支持一下嘛！老想着图小便宜的人也是做不了什么大事的。

大家如果看得起我，可以任意的转载和使用，但决不允许把本教程作为商业用途，只希望后来的兄弟少走一点弯路，我的目的也就达到了。

另外如果您在实际的产品开发中有什么问题的话，可以给我来电或发 E-mail：jiguo@citiz.net 联系，我会尽我的所能帮助大家。谢谢!

附录一

MCS-51 单片机指令表

助 记 符	指 令 说 明	字节数	周期数
（数 据 传 递 类 指 令）			
MOV A，Rn	寄存器传送到累加器	1	1
MOV A，direct	直接地址传送到累加器	2	1
MOV A， @Ri	累加器传送到外部 RAM(8 地址)	1	1
MOV A，#data	立即数传送到累加器	2	1
MOV Rn，A	累加器传送到寄存器	1	1
MOV Rn，direct	直接地址传送到寄存器	2	2
MOV Rn，#data	累加器传送到直接地址	2	1
MOV direct，Rn	寄存器传送到直接地址	2	1
MOV direct，direct	直接地址传送到直接地址	3	2
MOV direct，A	累加器传送到直接地址	2	1
MOV direct，@Ri	间接 RAM 传送到直接地址	2	2
MOV direct，#data	立即数传送到直接地址	3	2
MOV @Ri，A	直接地址传送到直接地址	1	2
MOV @Ri，direct	直接地址传送到间接 RAM	2	1
MOV @Ri，#data	立即数传送到间接 RAM	2	2
MOV DPTR，#data16	16 位常数加载到数据指针	3	1
MOVC A，@A+DPTR	代码字节传送到累加器	1	2
MOVC A，@A+PC	代码字节传送到累加器	1	2
MOVX A，@Ri	外部 RAM(8 地址)传送到累加器	1	2
MOVX A，@DPTR	外部 RAM(16 地址)传送到累加器	1	2
MOVX @Ri，A	累加器传送到外部 RAM(8 地址)	1	2
MOVX @DPTR，A	累加器传送到外部 RAM(16 地址)	1	2
PUSH direct	直接地址压入堆栈	2	2
POP direct	直接地址弹出堆栈	2	2
XCH A,Rn	寄存器和累加器交换	1	1
XCH A, direct	直接地址和累加器交换	2	1
XCH A, @Ri	间接 RAM 和累加器交换	1	1
XCHD A, @Ri	间接 RAM 和累加器交换低 4 位字节	1	1
(算 术 运 算 类 指 令)			
INC A	累加器加 1	1	1
INC Rn	寄存器加 1	1	1
INC direct	直接地址加 1	2	1
INC @Ri	间接 RAM 加 1	1	1
INC DPTR	数据指针加 1	1	2
DEC A	累加器减 1	1	1
DEC Rn	寄存器减 1	1	1
DEC direct	直接地址减 1	2	2

助 记 符	指 令 说 明	字节数	周期数
DEC @Ri	间接 RAM 减 1	1	1
MUL AB	累加器和 B 寄存器相乘	1	4
DIV AB	累加器除以 B 寄存器	1	4
DA A	累加器十进制调整	1	1
ADD A,Rn	寄存器与累加器求和	1	1
ADD A,direct	直接地址与累加器求和	2	1
ADD A,@Ri	间接 RAM 与累加器求和	1	1
ADD A,#data	立即数与累加器求和	2	1
ADDC A,Rn	寄存器与累加器求和(带进位)	1	1
ADDC A,direct	直接地址与累加器求和(带进位)	2	1
ADDC A,@Ri	间接 RAM 与累加器求和(带进位)	1	1
ADDC A,#data	立即数与累加器求和(带进位)	2	1
SUBB A,Rn	累加器减去寄存器(带借位)	1	1
SUBB A,direct	累加器减去直接地址(带借位)	2	1
SUBB A,@Ri	累加器减去间接 RAM(带借位)	1	1
SUBB A,#data	累加器减去立即数(带借位)	2	1
(逻 辑 运 算 类 指 令)			
ANL A,Rn	寄存器“与”到累加器	1	1
ANL A,direct	直接地址“与”到累加器	2	1
ANL A,@Ri	间接 RAM“与”到累加器	1	1
ANL A,#data	立即数“与”到累加器	2	1
ANL direct,A	累加器“与”到直接地址	2	1
ANL direct, #data	立即数“与”到直接地址	3	2
ORL A,Rn	寄存器“或”到累加器	1	2
ORL A,direct	直接地址“或”到累加器	2	1
ORL A,@Ri	间接 RAM“或”到累加器	1	1
ORL A,#data	立即数“或”到累加器	2	1
ORL direct,A	累加器“或”到直接地址	2	1
ORL direct, #data	立即数“或”到直接地址	3	1
XRL A,Rn	寄存器“异或”到累加器	1	2
XRL A,direct	直接地址“异或”到累加器	2	1
XRL A,@Ri	间接 RAM“异或”到累加器	1	1
XRL A,#data	立即数“异或”到累加器	2	1
XRL direct,A	累加器“异或”到直接地址	2	1
XRL direct, #data	立即数“异或”到直接地址	3	1
CLR A	累加器清零	1	2
CPL A	累加器求反	1	1
RL A	累加器循环左移	1	1
RLC A	带进位累加器循环左移	1	1
RR A	累加器循环右移	1	1
RRC A	带进位累加器循环右移	1	1
SWAP A	累加器高、低 4 位交换	1	1

助 记 符	指 令 说 明	字节数	周期数
(控 制 转 移 类 指 令)			
JMP @A+DPTR	相对 DPTR 的无条件间接转移	1	2
JZ rel	累加器为 0 则转移	2	2
JNZ rel	累加器为 1 则转移	2	2
CJNE A,direct,rel	比较直接地址和累加器,不相等转移	3	2
CJNE A,#data,rel	比较立即数和累加器,不相等转移	3	2
CJNE Rn,#data,rel	比较寄存器和立即数,不相等转移	2	2
CJNE @Ri,#data,rel	比较立即数和间接 RAM,不相等转移	3	2
DJNZ Rn,rel	寄存器减 1,不为 0 则转移	3	2
DJNZ direct,rel	直接地址减 1,不为 0 则转移	3	2
NOP	空操作,用于短暂延时	1	1
ACALL add11	绝对调用子程序	2	2
LCALL add16	长调用子程序	3	2
RET	从子程序返回	1	2
RETI	从中断服务子程序返回	1	2
AJMP add11	无条件绝对转移	2	2
LJMP add16	无条件长转移	3	2
SJMP rel	无条件相对转移	2	2
(布 尔 指 令)			
CLR C	清进位位	1	1
CLR bit	清直接寻址位	2	1
SETB C	置位进位位	1	1
SETB bit	置位直接寻址位	2	1
CPL C	取反进位位	1	1
CPL bit	取反直接寻址位	2	1
ANL C,bit	直接寻址位“与”到进位位	2	2
ANL C，/bit	直接寻址位的反码“与”到进位位	2	2
ORL C,bit	直接寻址位“或”到进位位	2	2
ORL C，/bit	直接寻址位的反码“或”到进位位	2	2
MOV C,bit	直接寻址位传送到进位位	2	1
MOV bit，C	进位位位传送到直接寻址	2	2
JC rel	如果进位位为 1 则转移	2	2
JNC rel	如果进位位为 0 则转移	2	2
JB bit，rel	如果直接寻址位为 1 则转移	3	2
JNB bit，rel	如果直接寻址位为 0 则转移	3	2
JBC bit，rel	直接寻址位为 1 则转移并清除该位	2	2
（伪指令）			
ORG	指明程序的开始位置		
DB	定义数据表		
DW	定义 16 位的地址表		
EQU	给一个表达式或一个字符串起名		

DATA	给一个 8 位的内部 RAM 起名	
XDATA	给一个 8 位的外部 RAM 起名	
BIT	给一个可位寻址的位单元起名	
END	指出源程序到此为止	
（指令中的符号标识）		
Rn	工作寄存器 R0-R7	
Ri	工作寄存器 R0 和 R1	
@Ri	间接寻址的 8 位 RAM 单元地址（00H-FFH）	
#data8	8 位常数	
#data16	16 位常数	
addr16	16 位目标地址，能转移或调用到 64KROM 的任何地方	
addr11	11 位目标地址，在下条指令的 2K 范围内转移或调用	
Rel	8 位偏移量，用于 SJMP 和所有条件转移指令，范围-128～+127	
Bit	片内 RAM 中的可寻址位和 SFR 的可寻址位	
Direct	直接地址，范围片内 RAM 单元（00H-7FH）和 80H-FFH	
$	指本条指令的起始位置	

电路图我不再贴出来，大家可以直接到网站上去下载，网址 www.fj136.com/bbs/index.asp