迈航科技

单片机那些事儿

初级篇——数码管

残弈悟恩

2014

官方淘宝店铺：HTTP://SHOP109195762.TAOBAO.COM

郑重声明

本资料以残弈悟恩开发的 MGMC-V1.0 实验板为硬件平台，以残弈悟恩编写的《深入浅出玩转 51 单片机》为辅助教程，以残弈悟恩录制的《31 天环游单片机》为基础视频。

本资料以个人学习和工作的经验为素材，以单片机初学者、单片机项目开发者为对象。教大家如何走进单片机的世界，如何开发工程项目。限于时间和水平关系，资料中难免有过失之处，望各位高手批评指教，多多拍砖，拍累了，你们休息，我继续上路。

现已连载的方式免费共享于各大电子网站，供单片机新手们参考学习，可以自由下载传阅，但未经残弈悟恩允许，不得用于任何商业目的，若经发现，残弈悟恩将以愚公移山的精神追究到底。

版本：20140417（V1.0）　　制造者：残弈悟恩

让爱充满大地——花 1 秒时间，拯救 1 个人，传递 1 份爱

声明：只是残弈悟恩爱心的喷发，我得不到一分钱，各位不要多想，谢谢！

你知道吗？在非洲北边的某个地区，每一秒都有许许多多的人正在挨饿，每一天至少有一位儿童死于营养不足。你的一次点击就能让某位穷人得到 1.1 杯食物。当然你可以不相信有这样的链接或者是骗点击什么的。事实上，网站确实是帮穷人得 1.1 杯食物的，只要你点进去单机一下中间的黄色按钮，就会出来一系列介绍各种商品的网页（绝对免费的并且不会下载任何软件，也不会有电脑病毒），同时也会有人因为您的一次点击而得到 1.1 杯食物，食物是由商家提拱的，但爱心却是您献出的。如果你觉得残弈悟恩在忽悠大家，你不妨可以在网上查一下是真与假。

看到这本资料的朋友多数都是电子爱好者、单片机初学者，或者干电子这一行的，管你穷学生还是穷工人，只要能上网，只要愿花一秒种就可以了。人生在世，有两件事不能等：一、孝顺；二、行善。无论你是 LED 小灯、普通灯泡也好，还是荧光灯也吧，最重要就是要懂得用自身的光去照耀别人，光的强度并不重要。

点击链接：

http://www.thehungersite.com/clickToGive/home.faces?siteId=1&link=ctg_ths_home_from_ths_thankyou_sitenav

第五章　数码管

前面认识了 LED 和一些 C 语言的基本用法，接下来我们继续复习 C 语言知识和学习一种叫数码管的东西，顾名思义，数码管就是用管子显示数码。其实生活中随处都可以见到数码管，例如电梯里显示的楼层数，空调上显示的温度值等，都用到了数码管。如果玩转了单片机，其实显示这些东西都是很 easy 的事。

5.1 C 语言之数组、字符串

5.1.1 数组

一、数组的基本概念

我们在以后的程序编写中，要经常用到数组，那什么是数组，先来看看“官方”的定义和对数组的说明，之后残弈悟恩在来总结一下数组。

所谓数组，就是相同数据类型的元素按一定顺序排列的集合，就是把有限个类型相同的变量用一个名字命名，然后用编号区分它们变量的集合，这个名字成为数组名，编号成为下标。组成数组的各个变量称为数组的分量，也称为数组的元素，有时也称为下标变量。

从概念上讲，数组是具有相同数据类型的有序数据的组合，一般来讲，数组定义后满足以下三个条件。

（1）具有相同的数据类型；

（2）具有相同的名字；

（3）在存储器中是被连续存放的。

例如数组 ColArr[8]：

```
uChar8 code ColArr[8] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

该数组具有相同的数据类型 uChar8（unsigned char）；具有相同的名字 ColArr，无论数组加了关键词“code”将其存于 Flash 中，还是去掉“code”将数组存于 SRAM 中，它们都是存放在一块连续的存储空间中，其中存储器的知识详见第 3 章。

这里需要注意一点，数组有 8 个元素，但是数组的下标是从 0 开始一直到 7，这点一定要与实际生活的习惯（我们一般从 1 开始计数）区别开。这是一个一维数组，后面可能还会用到二维数组，甚至多维数组。这里以一维数组为主来做讲解，多维数组大家自行研究。

二、数组的声明和初始化

数组的声明很简单，其格式为：数据类型　数组名[数组长度]，例如：int Tab[10]。

说到初始化，若能确定数组的各个元素，数组当然可以在声明的时候直接初始化，例如：

```
uChar8 code ColArr[8] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

若数组声明时还不能确定其数组元素，而是以后要读读写写，这时可以先声明一个数组，并赋值 0，例如：int ArrLED[32]={0}，这时前面一定不能加修饰词“code”，读者说为何呢？若不知道，请在本书寻找答案。

这里需要注意一点，若已经给数组赋了所有的初值，即数组的元素已经确定，这时数组的长度可以省略，例如：char Arr[]={1,2,3}，数组的长度为 3，此时的 3 可以省略不写。

二、数组的使用和赋值

数组的使用要是用下标法来索取，那就很简单了，例如上面的 uChar8 ColArr[8]数组，直接可以将 ColArr[0]、ColArr[1]... ColArr[7]当做一个变量（或常量）赋值给想要操作的变量。其实前面的大部分程序都是这么做的，我就再不说了。倘若用指针操作，或许就有难度了，关于这点，等到笔记 13 讲解了指针再来阐述这个问题。

数组如何赋值，读者先自己想一想，之后请思考一个问题：下面三个选项，那种赋值是正确的？

定义两个数组：int OK[5]={1,2,3,4};　int ERR[5];

A）ERR = OK;　　B）ERR[5] = OK[5];　　C）ERR[5] = {1,2,3,4};

C 不支持把数组作为一个整体来进行赋值，也不支持用花括号括起来的列表进行赋值（初始化的时候除外）。这样一来，三种都不正确，望读者熟知。

5.1.2 字符串

字符串在液晶一章已经大量的应用过了，但或许读者对其与数组的区别理解的还不是太到位。所谓字符串就是以空字符（\0）结尾的 char 数组。由此定义可得出两条信息：

（1）字符串属于数组。　　（2）字符串末尾有个隐形的“\0”。

因此读者可以用操作数组的方法来操作字符串。对于字符串除了用数组的方式来操作以外，还可以用大量的库函数来操作字符串。

字符串的声明和赋值格式如：uChar8 code TAB1[]="^_^ Welcome ^_^ "，看上去和数组区别不大，但有个隐形的“\0”，意味着多了一种操作的方法，如：while(TAB1[i] !='\0')。

5.2 夯实基础——C 语言之函数

如何组织一个程序呢？在 C 语言的设计原则上把函数作为程序的构成模块。例如前面用过的库函数 printf()、_nop_()等，还有自己编写的 DelayMS()、Timer0Init()等函数。

5.2.1 什么是函数？

函数（function）是用于完成特定任务的程序代码的自包含单元。尽管 C 中的函数和其它语言中的函数、子程序或子过程等扮演着相同的角色，但是在细节上会有所不同。 某些函数会导致执行某些动作，比如 printf（）可使数据呈现在屏幕上；还有一些函数能返回一个值以供程序使用，如 strlen（）将制定字符串的长度传递给程序。一般来讲，一个函数可同时具备以上两种功能。

5.2.2 为什么使用函数？

第一，函数的使用可以省去重复代码的编写。如果程序中需要多次使用某种特定的功能，那么只需编写一个合适的函数即可。程序可以在任何需要的地方调用该函数，并且一个函数可以在不同的程序中调用，就像在许多程序中需要使用 DelayMS（）函数一样。

第二，即使某种功能在程序中只使用一次，将其以函数的形式实现也是有必要的，因为函数使得程序更加模块化，从而有利于程序的阅读、修改和完善。

5.2.3 函数的分类

一、从函数定义的角度看，函数可分为库函数和用户定义函数两种。

（1）库函数。系统提供，用户无需定义，也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。例如包含<stdio.h>之后就可以调用 printf（）函数。

（2）用户定义函数。由用户按需要写的函数。对于用户自定义函数，不仅要在程序中定义函数本身， 而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。

二、C 语言的函数兼有其它语言中的函数和过程两种功能，从这个角度看，又可把函数分为有返回值函数和无返回值函数两种。

（1）有返回值函数。此类函数被调用执行完后将向调用者返回一个执行结果，称为函数返回值。如数学函数即属于此类函数。由用户定义的这种要返回函数值的函数，必须在函数定义和函数说明中明确返回值的类型。

（2）无返回值函数。此类函数用于完成某项特定的处理任务，执行完成后不向调用者返回函数值。

三、从主调函数和被调函数之间数据传送的角度看又可分为无参函数和有参函数两种。

（1）无参函数。函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送。 此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。

（2）有参函数，也称为带参函数。在函数定义及函数说明时都有参数，称为形式参数(简称为形参)。在函数调用时也必须给出参数，称为实际参数(简称为实参)。进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。前面程序中见过的无参函数和有参函数就很多了，例如无参函数：Timer0Init(),有参函数：DelayMS（uInt16 ms)。

这里读者需要注意以下几点：

（1）实参可以是变量，也可以是表达式，或者是最直接的值。目的都是把实参的值传递给自定义函数中的形参。我就不举例了，以后碰到了再说明。

（2）函数的值只能通过 return 语句返回主调函数。return 语句一般形式为：return 表达式，或者为：return（表达式)。该语句的功能是计算表达式的值，并返回给主调函数。在函数中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，因此只能返回一个函数值。

（2）函数值的类型和函数定义中函数的类型应保持一致。

（3）不返回函数值的函数，可以明确定义为“空类型”，类型说明符为“void”。

接下来举个例子，综合说明一下函数的执行和各个参数的传递过程。程序源码如下。

```
1.  char AddXYZ(char a,char b,char x);
2.  void main(void)
3.  {
```

```
4.          char x = 10;
5.          char y = 20;
6.          char z = 30;
7.          char xyz = 0;
8.          xyz = AddXYZ(x,y,z);
9.    }
10.   char AddXYZ(char a,char b,char c)
11.   {
12.         char abc = 0;
13.         abc = a + b + c;
14.         return(abc);
15.   }
```

程序说明：第 1 行，由于函数 AddXYZ()定义于主函数的后面，所以需要声明此函数，当然若定义在主函数前就不需要声明了；第 8 行，调用函数 AddXYZ()，并且将实参 x(10)、y(20)、z(30)传递给形参 a、b、c，之后运行第 13 行程序，计算 a + b + c，并将值赋值给 abc，接着执行第 14 行程序，将 abc 返回，再赋值给 xyz，这样 xyz 的值就为：60。剩余的关于函数的编码风格之类的请读者去拜读一下《C 语言深度解剖》第六章，写的确实很好，堪称绝品。

5.2.4 函数的命名规则

其实说到函数的命名，就没有变量那么多规则了，没规则不意味着函数的命名可以随便。例如计算两个数的和，若写成这样的函数 jia(int x,int y)，或许读者现在能明白，那你能保证过两周、两个月后还能就只看函数名就知道该函数的功能吗？因此函数的命名还是一样，望文知义很重要，书写格式很重要。

前面在讲述变量命名时说过，变量的命名只要有两种方法：匈牙利命名法和驼峰大小式命名法。说到这里，就我个人认为，函数的命名主要是利用驼峰大小写中的大驼式命名，例如：MyFirstName、WrDataToLCD。

同样在讲述变量命名时最后我补充了两点，其中一点是说变量命名使用名词性词组，一般结构为：目标词 + 动词（的过去分词）+ [状语] + [目的地]。例如 DataGotFromSD、DataDeletedFromSD。那函数如何命名呢？

函数的格式稍微有变，先给读者来两个例子，以便做对比。例如：GetDataFromSD、DeleteDataFromSD，两者大致意思都是从 SD 中取得、删除数据，但结构视乎有别，现将函数的命名一般结构总结为：动词（一般现在时） + 目标词 + [状语] + [目的地]。OK，函数的命名就先说这么多，更好的就留读者自行研究了。

5.3 夯实基础—C 语言之指针

提到指针，或许包括残弈悟恩在内的大部分读者都很“过敏”，因为觉得指针难，就现在来说，我还是对指针了解的不够，但我还是想把它学会、用好，同时抛砖引玉地讲好，这只能是个目标了。

究竟什么叫做指针？一般来将，指针是一个其数值为地址的变量（或更一般地说是一个数据对象）。正如 char 类型的变量用字符作为其数值，而 int 类型变量的数值是整数，指针

变量的数值表示的是地址。

5.3.1 小试牛刀——指针

如果读者您将某个指针变量命名为 ptr，就可以使用如下语句：

```
prt = & pooh;        /*  把 pooh 的地址赋给 ptr */
```

对于这个语句，我们称 ptr“指向”pooh。ptr 和&pooh 的区别在与前者为一变量，而后者是一个常量。当然，ptr 可以指向任何地方。如：ptr = & abc，这时 ptr 的值是 abc 的地址。

要创建一个指针变量，首先需要声明其类型。这就需要下面介绍的新运算符来帮忙了。

假如 ptr 指向 abc，例如：ptr = & abc，这时就可以使用间接运算符“*”（也称作取值运算符）来获取 abc 中存放的数值（注意与二元运算符*的区别）。

```
val = * ptr;          /*  得到 ptr 指向的值  */
```

这样就会有：val = abc。由此看出，使用地址运算符和间接运算符可以间接完成上述语句的功能，这也正是“间接运算符”名称的由来。因此就有了我们常听到的：所谓的指针就是用地址去操作变量。

5.3.2 指针的声明

前面章节，读者们用了大量的基本变量，同时也掌握了变量的声明。那应该如何声明指针变量呢？或许读者会这样声明一个指针：pointer ptr，我告诉读者，这样声明一个指针变量是不正确的。

为什么不能这样声明？因为这对于声明一个变量为指针是远远不够的，还需要说明指针所指向变量的类型。原因是不同的变量类型占用的存储空间大小不同，而有些指针操作需要知道变量类型所占用的存储空间。同时，程序也需要了解地址中存储的是哪种数据。例如，long 和 float 两种类型的数值可能使用相同大小的存储空间，但是它们的数据存储方式完全不同。指针的声明形式如下：

```
int * abc;                /* abc 是只需一个整数变量的指针  */
float * bcd, * cde;       /* bcd 和 cde 是指向浮点变量的指针  */
```

读到这里，每位读者都知道，上面的 abc、bcd 等都是一个定义的指针，那这个指针究竟是一个什么东西，或者说在内存中占多大的空间，那我们用 sizeof 测试一下（32 位系统）：sizeof(abc)、sizeof(bcd)，它们的值都为：4。

这说明了什么？说明一个基本的数据类型（包括结构体等自定义类型）加上“*”号就构成了一个指针类型的变量，这个变量的大小是一定的，与“*”号前面的数据类型无关。“*”号前面的数据类型只是说明指针所指向的内存里存储的数据类型。所以 32 位系统下，不管什么样的指针类型，其大小都为 4byte。读者当然可以测试一下 sizeof(void*)。

5.3.3 指针与数组的藕断丝连

关于数组残弈悟恩前面简简单单讲述了一下，不知读者发现了没有，在有些例程中，用下标法操作数组比较麻烦，若用指针来操作数组，或许能起到事半功倍的效果，至少有些时候比较方便、快捷，具体实例，我们以后遇到了在来讲述。那么接下来我们看看这两者之间的恩恩怨怨。

一、叙叙数组

这里定义一个数组：int a[5]，所有人都明白定义了一个数组，其包含了 5 个 int 型的数据，可以用 a[0]、a[1]等来访问数组里面的每一个元素，那么这些元素的名字就是 a[0]、a[1]...吗？先看下面的示意图（如图 5-1 所示），再来阐述这个问题。

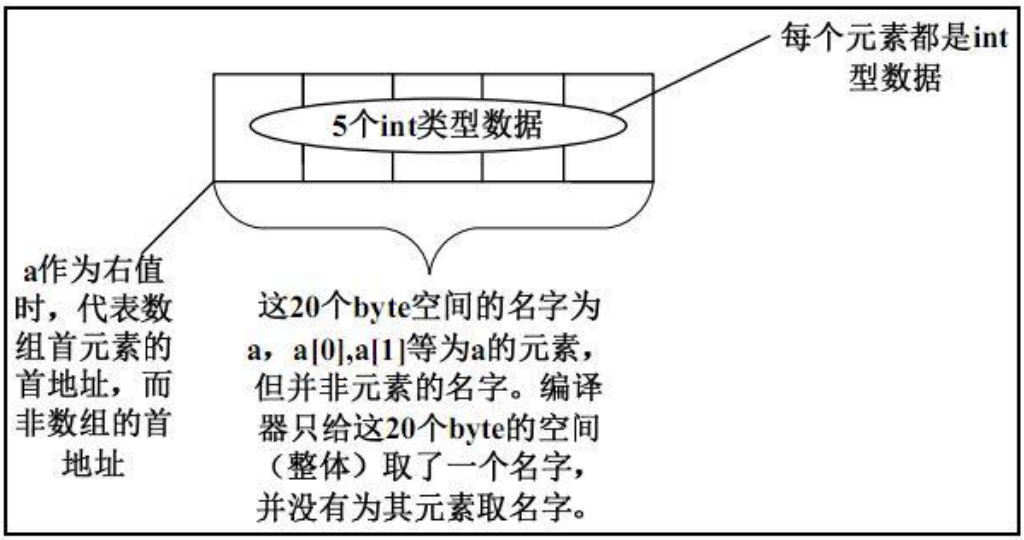

图 5-1 数组示意图

如上图所示，当定义了一个数组 a 时，编译器根据指定的元素个数和元素的类型分配确定大小（元素类型大小×元素个数）的一块内存，并把这块内存的名字命名为 a。名字 a 一旦与这块内存匹配就不能改变。a[0]、a[1]等为 a 的元素，但并非元素的名字。数组的每一个元素都是没有名字的。读者是否还记得，残弈悟恩在笔记 10 的夯实基础部分还留下了两个问题，那现在就来回答那三个问题吧。

sizeof(OK)的值为 sizeof(int)*5，32 位系统下位 20。

sizeof(OK[0])的值为 sizeof(int)，32 位系统下位 4。

sizeof(OK[5])的值在 32 位系统下位 4。这里并没有出错，为什么呢？因为 sizeof 是关键字，不是函数。函数求值是在运行的时候，而关键字 sizeof 求值是在编译的时候。虽然并不存在 OK[5]这个元素，但是这里也并没有去真正访问 OK[5]，而是仅仅根据数组元素的类型来确定其值，所以这里使用 OK[5]并不会出错。

现在回过头来继续讲解上面的数组 a[5]。sizeof(&a[0])的值在 32 位系统下位 4，这个很好理解，意思是取元素 a[0]的首地址。sizeof(&a)的值在 32 位系统下也为 4，意思当然是取数组 a 的首地址。

二、省政府和市政府的区别——&a[0]和&a 的区别

这里&a[0]和&a 到底有什么区别呢？a[0]是一个元素，a 是整个数组，虽然&a[0]和&a 的值一样，但其意义不一样。前者是数组首元素的首地址，而后者是数组的首地址。举个例子：甘肃省的省政府在兰州，而兰州市的市政府也在兰州。两个政府都在兰州，但其代表的意义完全不同。这里也是同一个意思。

三、数组名 a 作为左值和右值的区别

简单而言，出现在赋值符“=”右边的就是右值，出现在赋值符“=”左边的就是左值。比如：a = b，则 a 为左值，b 为右值。

（1）当 a 作为右值时其意义与&a[0]是一样，代表的是数组首元素的首地址，而不是数

组的首地址。但注意这仅仅是一种代表。

（2）a 不能作为左值。当然可以将 a[i]当做左值，这时就可以对其操作了。

四、数组与指针

读者需要注意，我将数组和指针放到这里讲解，是为了区分它们，而不是为了将它们联系起来。请读者铭记：数组就是数组，指针就是指针。它们是完全不同的两码事！它们之间没有任何关系，只是经常穿着相似的衣服迷惑读者罢了。

1.以指针的形式访问和以下标的形式访问

在函数内部有两个定义：A. char *p = “abcdef”; B. char a[] = “123456”;

（1）以指针的形式访问和以下标的形式访问指针

例子 A 定义了一个指针变量 p，p 本身在栈上占 4 个 byte，p 里存储的是一块内存的首地址。这块内存在静态区，其空间大小为 7 个 byte，这块内存也没有名字。对这块内存的访问完全是匿名的访问。比如现在需要读取字符‘e’，我们有两种方式：

1）以指针的形式：*(p+4)。先取出 p 里存储的地址值，假设为 0x0000FF00，然后加上 4 个字符的偏移量，得到新的地址 0x0000FF04。然后取出 0x0000FF04 地址上的值。

2）以下标的形式：p[4]。编译器总是把以下标的形式的操作解析为以指针的形式的操作。p[4]这个操作会被解析成：先取出 p 里存储的地址值，然后加上中括号中 4 个元素的偏移量，计算出新的地址，然后从新的地址中取出值。也就是说以下标的形式访问在本质上与以指针的形式访问没有区别，只是写法上不同罢了。

（2）以指针的形式访问和以下标的形式访问数组

例子 B 定义了一个数组 a，a 拥有 7 个 char 类型的元素，其空间大小为 7。数组 a 本身在栈上面。对 a 元素的访问必须先根据数组的名字 a 找到数组首元素的首地址，然后根据偏移量找到相应的值。这是一种典型的“具名+匿名”访问。比如现在需要读取字符‘5’，我们有两种方式：

1）以指针的形式：*(a+4)。a 这时候代表的是数组首元素的首地址，假设为 0x0000FF00，然后加上 4 个字符的偏移量，得到新的地址 0x0000FF04。然后取出 0x0000FF04 地址上的值。

2）以下标的形式：a[4]。编译器总是把以下标的形式的操作解析为以指针的形式的操作。a[4]这个操作会被解析成：a 作为数组首元素的首地址，然后加上中括号中 4 个元素的偏移量，计算出新的地址，然后从新的地址中取出值。

由上面的分析，我们可以看到，指针和数组根本就是两个完全不一样的东西。只是它们都可以“以指针形式”或“以下标形式”进行访问。一个是完全的匿名访问，一个是典型的具名+匿名访问。一定要注意的是这个“以 XXX 的形式的访问”这种表达方式。

另外一个需要强调的是：上面所说的偏移量 4 代表的是 4 个元素，而不是 4 个 byte。只不过这里刚好是 char 类型数据 1 个字符的大小就为 1 个 byte。记住这个偏移量的单位是元素的个数而不是 byte 数，在计算新地址时千万别弄错喔！

好吧，关于指针和数组就先说这么，其实这两者之间的知识点远不止这些，限于篇幅，我就先引玉这么多了，剩下的读者自行研究吧。

5.3.4 指针与函数

说到指针和函数，内容当然少不到哪儿去。这里以指针如何在函数间通信为例，来说说指针的在函数中的作用。具体源码如下：

```
1.  #include <stdio.h>
2.  void interchange(int * u,int * v);
```

```
3.  int main(void)
4.  {
5.      int x = 5,y = 10;
6.      printf("Originally x = %d and y = %d.\n",x ,y);
7.      interchange(&x,&y);     /* 向函数传送地址 */
8.      printf("Now x = %d and y = %d.\n",x,y);
9.      return 0;
10. }
11. void interchange(int * u, int * v)
12. {
13.     int temp;
14.     temp = *u;     /* temp 得到 u 指向的值 */
15.     *u = *v;
16.     *v = temp;
17. }
```

现对此例程作简要解释。由第 7 行可以看出，函数传递的是 x 和 y 的地址而不是它们的值。这就意味着 interchange（）函数原型声明和定义中的形式参数 u 和 v 将使用地址作为它们的值。因此，它们应该声明为指针。由于 x 和 y 都是整数，所以 u 和 v 是指向整数的指针，因而有了 11 行所示的函数声明；第 14 行，因为 u 的值是&x，所以 u 指向 x 的地址。这就意味着*u 代表了 x 的值，而这正是我们需要的数值，一定不要写成：temp=u，因为赋值给变量 temp 的值是 x 的地址而不是 x 的值，所以不能实现数值的交换。这时读者再回过头去看看实例 18 的 75 行，是不是也用指针作为桥梁呢

5.4 夯实基础——C 语言之结构体

《C Primer Plus》中有这么一句话：设计程序最重要的一个步骤就是选择一个表示数据的好方法。那简单的变量、复杂的数值好吗？其实在多数程序设计中，使用这些都是不够的。鉴于这种情况，C 使用结构变量进一步增强了表示数据的能力。

说到结构体，残弈悟恩实在忍不住想夸 ST 公司为 STM32 编写的函数库，因为里面对结构体的应用实在让我佩服。或许是我的见识短，或者是高手写的程序拜读的不多。我写这个就是希望读者能从玩单片机的时候开始，好好掌握结构体，这样以后或许对玩 ARM 很有帮助。

5.4.1 结构体

结构体(struct)是由一系列具有相同类型或不同类型的数据构成的数据集合，也叫结构。既然上面说了很牛 X，那就来学学它又有何妨呢？

一、结构体的声明

结构声明是描述结构如何组合的主要方法。一般声明并定于变量的方式有以下两种。

第一种	第二种
struct book{ char title[MAXTITL]; char author[MAXAUTL]; float value;	struct book{ char title[MAXTITL]; char author[MAXAUTL]; float value;

}; struct book library;	} library;

先以第一种为例来对其结构体做简要介绍。该声明描述了一个由两个字符数组和一个 float 变量组成的结构。它并没有创建一个实际的数据对象，而是描述了组成这类对象的元素（有时候可以将结构声明叫做模板）。首先使用关键字 struct，它表示接下来时一个结构。后面是一个可选的标记（book），它是用来引用该结构的快速标记。例如后面定义的 struct book library，意思是把 library 声明为一个使用 book 结构设计的结构变量。

在结构声明中，接下来是用一对花括号括起来的结构成员列表。每个成员变量都用它自己的声明来描述，用一个分号来结束描述。例如，title 是一个拥有 MAXTITL 组成的元素的 char 数组。每个成员可以是任何一种 C 的数据类型，甚至可以是其他结构。

标记名（book）是可选的。但是在用如第一种方式建立结构（在一个地方定义结构设计，而在其它地方定义实际的结构变量）时，必须使用标记。若没有标记名，则称之为无名结构体。

结合上面两种方式，我们可以得出这里的“结构”有两个意思。一个意思是“结构设计”，例如对变量 title、author 的设计就是一种结构设计。另一层意思应该是创建一个“结构变量”，例如若定义的 library，就是创建一个变量很好的举证。其实这里的 struct book 所起的作用就像 int 或 float 在简单声明中的作用一样。

二、结构体变量的初始化

结构是一个新的数据类型，因此结构变量也可以像其它类型的变量一样赋值、运算，不同的是结构变量以成员作为基本变量。

结构成员的表示方式为：结构变量.成员名。

这里的“.”是成员（分量）运算符，它在所有的运算中优先级最高，因此将“结构变量.成员名”可以看成是一个整体，则这个整体的数据类型与结构中该成员的数据类型相同，这样就可像前面所讲的变量那样使用。例如读者可以对以上述所举例子做以下的赋值操作。

library.title = “深入浅出玩转 51 单片机”;library.author = “残弈悟恩”;

当然也可以边声明结构体，边初始化结构体。

```
struct student{
    long int num;
    char name[20];
    char address[30];
}a = {123456,“残弈悟恩”,“123 HuiNing Road”};
```

三、结构体的数组

结构数组就是具有相同结构类型的变量集合。上面讲解到一个结构体变量中可以存放一组数据（如一个学生的学号、姓名、家庭地址等数据）。如果有 10 个学生的数据需要参与运算，显然应该用数组，这就是结构体数组的由来。结构体数组与以前介绍过的数据值型数组不同之处在于每个数组元素都是一个结构体类型的数据，它们分别包括各个成员（分量）项。

接着上面声明的结构体：student，在来定义一个结构体数组：struct student stu[10];这样就可以用类似于操作二维数组的方式对其赋值、运算了，限于篇幅，这里就不举例了。

四、指向结构体变量的指针

一个结构体变量的指针就是该变量所占据的内存段的起始地址，可以设一个指针变量，用来指向一个结构体变量，此时该指针变量的值是结构体变量的起始地址。指针变量也可以

用来指向结构体数组中的元素。

再以上面“student”结构体为例，来定义一个结构体变量和结构体指针：

struct student stuA;　struct student *p;

接着让指针 p 指向 stuA，则有：p=&stuA；这样就可以对成员 num 做这样的赋值操作：stuA.num = 123456 或者(*p).num = 123456。别的都好理解，这里读者需要注意*p 两侧的括号不可省略，因为成员运算符“.”优先于“*”运算符，若取了括号则：*p.num 就等价于*(p.num)，这显然不合题意。

还好，在 C 语言中，为了使用方便和使之直观，可以把(*p).num 改用 p->num 来代替，它表示 p 指向结构体变量中的 num 成员。

这样，结构体的成员变量访问就有三种方式，分别为：a.结构体变量.成员名；b.(*p).成员名；c.p->成员名。

5.4.2 枚举

在实际应用中，有的变量只有几种可能的取值。如人的性别只有两种（除人妖）可能的取值，星期只有七种可能的取值。在 C 语言中对这样取值比较特殊的变量可以定义为枚举类型。所谓枚举是指将变量的值一一列举出来，变量只限于列举出来的值的范围内取值。

一、枚举的定义

定义一个变量是枚举类型，可以先定义一个枚举类型名，然后再说明这个变量是该枚举类型。例如：enum weekday{sun,mon,tue,wed,thu,fri,sat};

定义了一个枚举类型名 enum weekday，然后定义变量为该枚举类型。例如：enum weekday day; 当然，也可以直接定义枚举类型变量。例如：enum weekday{sun,...,sat} day;

其中 sum、mon、…、sat 等称为枚举元素或枚举常量，它们是用户定义的标识符。

二、关于枚举的几点说明

（1）枚举元素不是变量，而是常数，因此枚举元素又称为枚举常量。因为是常量，所以不能对枚举元素进行赋值。

（2）枚举元素作为常量且它们是有值的，C 语言在编译时按定义的顺序使它们的值为：0、1、2、…。

枚举定义以后，默认情况下，值是从 0 开始，按顺序依次加 1。若有赋值语句 day=mon;则 day 变量的值为 1。当然，这个变量值是可以输出的。例如：printf("%d",day);将输出整数 1。

如果在定义枚举类型时指定元素的值，也可以改变枚举元素的值。例如：enum weekday{sun=7,mon=1,tue,wed,thu,fri,sat}day;这时 sun 为 7，mon 为 1，以后元素顺次加 1，所以 sat 就是 6 了。

（3）枚举值可以用来作判断。例如：if (day == mon){…}、if (day > mon){…}。枚举值的比较规则是：按其在声明时的顺序号比较，如果说明时没有为其指定值，则第一个枚举元素的值认作 0，从而由 mon > sun、sat > fri。

（4）一个整数不能直接赋给一个枚举变量，必须强制进行类型转换才能赋值。例如：day = (enum weekday)2;这个赋值的意思是将顺序号为 2 的枚举元素赋给 day，相当于 workday = tue;

三、枚举与#define 宏的区别

1）#define 宏常量是在预编译阶段进行简单替换；枚举常量则是在编译的时候确定其值。

2）一般在编译器里，可以调试枚举常量，但是不能调试宏常量。

3）枚举可以一次定义大量相关的常量，而#define 宏一次只能定义一个。

5.4.3 大刀阔斧——typedef

一、typedef 与结构体形影不离

typedef 的用途当然不止现在讲解的这么一点，无论多少，先说说其在结构体定义中的一小点知识。先来举个例子。

```
typedef struct complex{
    float real;
    float imag;
}COMPLEX;
```

这样您就可以用类型 COMPLEX 代替 struct complex 来表示复数。使用 typedef 的原因之一是为经常出现的类型创建一个方便的、可识别的名称。因此这里的意思就是将 struct complex 命名为 COMPLEX，也即给 struct complex 起了一个别名“COMPLEX”。

使用 typedef 来命名一个结构类型时，可以省去结构的标记，例如：

```
typedef struct{double x;double y;}rect;
```

假设这样使用 typedef 定义的类型名：rect r1 = {3.0,5.0};rect r2;r2 = r1;这就可以被“翻译”成：

```
struct {double x; double y;} r1 = {3.0,5.0};
struct {double x; double y;} r2;  r2 = r1;
```

如果两个结构的声明都不使用标记，但是使用同样的成员（成员名和类型都匹配），那么 C 认为这两个结构具有同样的类型，因此将 r1 赋值给 r1 是一个正确的操作了。

再如：

```
typedef enum workday{
saturday,sunday = 0,monday,tuesday,wednesday, thursday,friday
}workday;        //此处的 workday 为枚举型 enum workday 的别名
workday today,tomorrow;
```

这样变量 today 和 tomorrow 的类型为枚举型 workday，也即 enum workday。

二、typedef 与#define 的区别

读者铭记一句话：typedef 的真正意思是给一个已经存在的**数据类型**（注意：是**类型**不是变量）取一个别名，而非定义一个新的数据类型。

我在定义 uChar8、uInt16（声明、宏定义、起别名）时有点凌乱，对不住读者了，^_^。残弈悟恩不高兴了是：#define uChar8 unsigned char，高兴了又是：typedef unsigned char uChar8;它们那个好，有区别吗？暂时的答案是没区别，但别高兴的太早，这里的没区别并不代表哪儿都没区别，更不能代表以后的随便“滥用职权”。

（1）问题一：A.#define Char8 char
unsigned Char8 i = 20;

B.typedef char Char8;
unsigned Char8 j = 10;

为何 A 是正确的，而 B 是错误的吗？对于 A 来说，因为在预编译的时候 Char8 直接被

char 替换，那么 unsigned Char8 i 就等价于 unsigned char i，这样当然是对的。那么对于 B 呢？上面说过，它只是起个别名，不支持这种数据类型的扩展，因为错误是难免的。

（2）问题二：C.#define PCHAR char*　　D.typedef char* pchar;

PCHAR p1,p2;　　pchar p3,p4;

两组代码编译都没有问题，但是，这里的 p2 还是指针吗？答案是：NO 的严重，而仅仅是一个 char 类型的字符。所以用#define 和 typedef 的时候要慎之又慎啊。

在应用 typedef 是要注意两点：

1）typedef 只是给现有的数据类型起了一个别名，更不同于宏，也不是简单的字符串替换。例如定义：typedef char* PSTR;则：const PSTR 并不是 const char*!

2）typedef 在语法上是一个存储类的关键字（auto、extern、static），但它并不真正影响对象的存储特性。例如：typedef static char Char8;//肯定是不可行的

5.5 数码管的点点滴滴

5.5.1 原理说明

（1）数码管分类：

数码管是一种半导体发光器件，也称半导体数码管，它是将若干发光二极管按一定图形排列并封装在一起的最常用的数码管显示器件之一。LED 数码管具有发光显示清晰、响应速度快、省电、体积小、寿命长、耐冲击、易于各种驱动电路连接等优点，在各种数显仪器仪表、数字控制设备中得到广泛应用。

数码管按段数分为七段数码管和八段数码管，八段数码管比七段数码管多了一个发光二极管单元（多一个小数点显示），按能显示多少个“8”可分为 1 位、2 位、3 位、4 位等等。按发光二极管单元连接方式分为共阳极数码管和共阴极数码管。共阳极数码管是指将数码管在应用时应将公共极 COM 接到+5V，当某一字段发光二极管的阴极为低电平时，相应字段就点亮，当某一字段的阴极为高电平时，相应字段就不亮。共阴极数码管是指所有发光二极管的阴极接到一起形成共阴极（COM）的数码管，共阴极数码管在应用时应将公共极 COM 接到地线 GND 上，当某一字段发光二极管的阳极为高电平时，相应字段就点亮，当某一字段的阳极为低电平时，相应字段就不亮。

（2）数码管的结构及特点

目前，常用的小型 LED 数码管多为“8”字形数码管，它内部由 8 个发光二极管组成，其中 7 个发光二极管（a~g）作为 7 段笔画组成“8”字结构（故也称 7 段 LED 数码管），剩下的 1 个发光二极管（h 或 dp）组成小数点，如图 5-2 所示。各发光二极管按照共阴极或共阳极的方法连接，即把所有发光二极管的负极或正极连接在一起，作为公共引脚。而每个发光二极管对应的正极或者负极分别作为独立引脚（称“笔段电极”），其引脚名称分别与图中的发光二极管相对应。

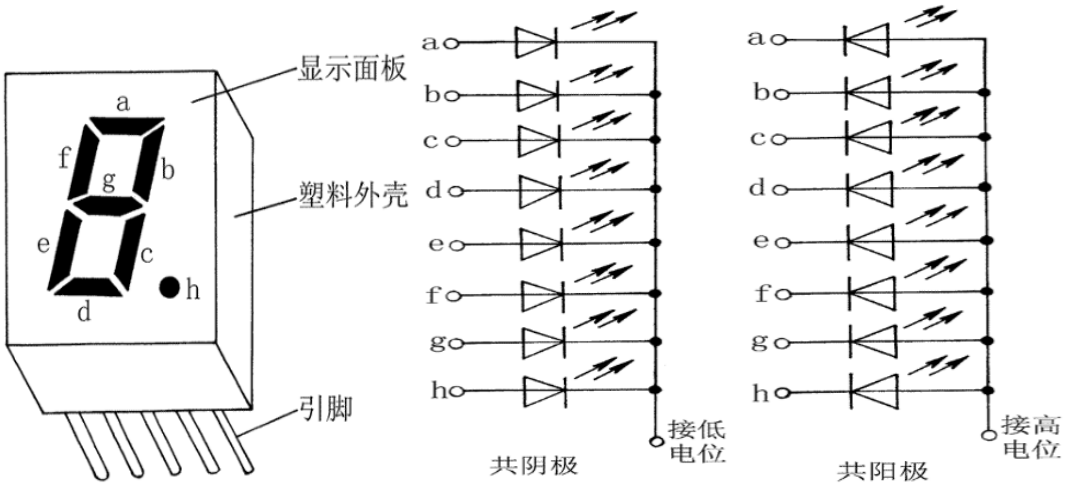

图 5-2　数码管结构示意图

（3）数码管简易测试方法

一个质量保证的 LED 数码管，其外观应该是做工精细、发光颜色均匀、无局部变色及无漏光等。对于不清楚性能好坏、产品型号及管脚排列的数码管，可采用下面介绍的简便方法进行检测。

1）干电池检测法

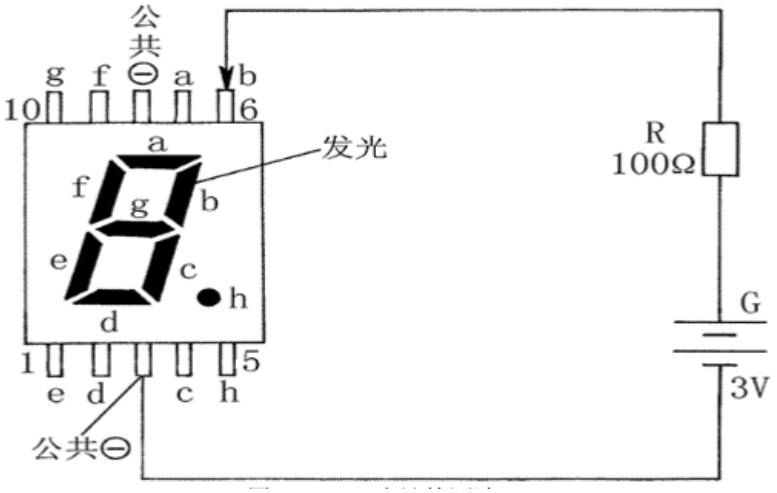

图 5-3　干电池测试法

取两节普通 1.5V 干电池串联（3V）起来，并串联一个 100 欧姆、1/8W 的限流电阻器，以防止过电流烧坏被测 LED 数码管。将 3V 干电池的负极引线接在被测数码管的公共阴极上，正极引线依次移动接触各笔段电极。当正极引线接触到某一笔段电极时，对应的笔段就发光显示。用这种方法就可以快速测出数码管是否有断笔或连笔，并且可相对比较出不同的笔段发光强弱是否一致。若检测共阳极数码管，只需将电池的正、负极引线对调一下即可。

2）万用表检测法

使用万用表的二极管档或者使用“R×10k”电阻档，检测方法同干电池检测法，残弈悟恩上大学时一直用万用表来测，因为干电池测试比较麻烦，万用表测试方便、快捷。

5.5.2 硬件设计

前面提到了单片机的拉电流比较小（100~200uA），灌电流比较大（最大是 25mA，但残弈悟恩推荐别超过 10mA），直接用来驱动数码管肯定是不行的，所以扩流电路是必须的，如

果用前面讲述的三级管来驱动，在原理上是正确无误的，可是 MGMC-V1.0 实验板上的单片机只有 32 个 I/O 口，可板子又外接了好多器件，这样下来 I/O 口根本不够用，所以得想个两全其美的方法，即扩流又扩 I/O 口，综合考虑之下，选用 74HC573 锁存器来解决这两个问题，其实以后做工程时，若用到数码管，三极管、锁存器这两种方案都不是太好，因为这种方法是要靠 CPU 不断地刷新来显示，而工程中，CPU 还有好多事要干（既要当爹、又要当妈），所以这种不方案不可取，取而代之的是专门的集成 IC，如 FD650 就具有数码管驱动和按键扫描功能的 IC，想改变数据或者读取按键值时，只需操作该芯片就 OK 了，这样可以大大提高了 CPU 的利用效率。接下来看一看 MGMC-V1.0 实验板上数码的硬件设计电路图，如图 5-4 所示。

COM0~COM7 分别接单片机的 P0.0~P0.7，WE 和 DU 分别接单片机的 P1.6、P1.7。首先解释一下什么是段选，什么位选。残弈悟恩总结为：段选是亮什么（例如亮 0 还是 1、2…），位选是那个亮（第 1 位亮，还是第 5 位亮），是送段选还是送位选分别由 U6、U5 控制。

拓展 74HC573。考虑到有些读者还没有学数电，这里就不从原理上来讲解，而是形象的说明几句。74HC573 说难也不简单，说简单也不难，无论读者是否学过数电，读者只需将 573 理解为一扇大门（区别是这个门是单向的），其中第 11 管脚控制着其门的“开”、“关状态，高电平——大门敞开，低电平-大门关闭。还有 D0~D7 是进，Q0~Q7 为出。知道这么多完全足够了，若想从原理上深入理解，读者可以去看看 74HC573 的数据手册（DataSheet）。

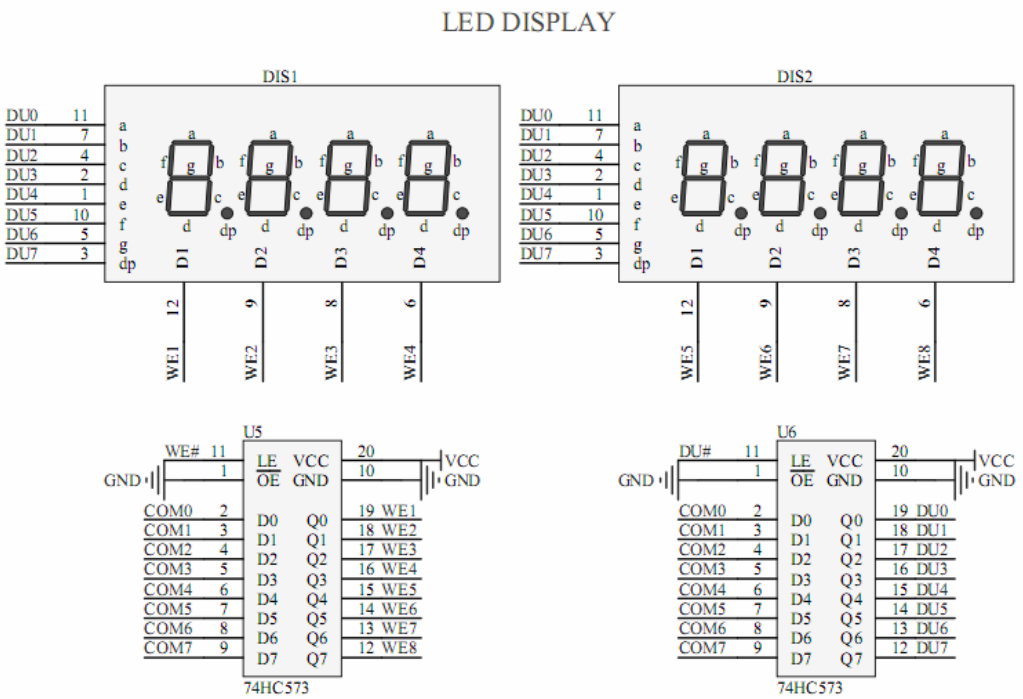

图 5-4　数码管驱动电路

5.5.3 软件分析

好，搞定了数码管的硬件，接着看看软件如何实现，例如想让 8 个数码管都亮“1”，那该如何操作呢？

要让 8 个都亮，那意味着“位选”全部选中，不难理解吧，MGMC-V1.0 开发板用的是共阴极数码管，要选中哪一位，只需给每个数码管对应的位选线上送低电平，即 U5 输出端为：0b0000 0000；若是共阳极，相反给高电平。那又如何亮“1”，由于是共阴极数码管，所以

段选高电平有效（即发光二极管阳极为“1”，阴极为“1”，不亮都难），亮“1”等价于 b、c 段亮，别的全灭，这样只需 U6 的输出端电平为：0b0000 0110（注意高位在前，低位在后）。读者再算一算，亮“3”的编码是不是：0x4f，亮“7”的编码是不是：0x7f”。这里需要读者注意，给数码管的段选数据、位选数据都是由 P0 口给的，只是在不同的时间给的对象不同，并且给对象给的数据也不同。举例说明，读者的手即可以写字，又可以吃饭，还可以打篮球。只是在上课的时候手上拿的是笔，而吃饭时拿的又是筷子，打球时手上拍的篮球。概况的讲，就是所谓的时分复用。这样，就可以写出如下的数码管驱动程序。最后显示效果如图 5-5 所示。

```
1.  #include <reg52.h>
2.  sbit SEG_SELECT = P1^7;
3.  sbit BIT_SELECT = P1^6;
4.  void main(void)
5.  {
6.      BIT_SELECT = 1;          //开位选的大门
7.      P0 = 0x00;               //让位选数据通过（选中 8 位）
8.      BIT_SELECT = 0;          //扔下笔，准备抱篮球
9.      SEG_SELECT = 1;          //开段选的大门
10.     P0 = 0x06;               //送段选数据过去（亮“1”的编码）
11.     SEG_SELECT = 0;          //扔下篮球，以便于拿筷子
12.     while(1);
13. }
```

图 5-5　8 位数码管显示数字 1

再来总结一下整个点亮“1”的过程，当理解了时分复用以后，这个问题就变得简单了。我发现在培训时，好多同学还是不能理解，那就再举个例子，有两个读者张三（U5）、李四（U6），共用一个残弈悟恩（单片机 P0 口），先开我与张三的通信开关（BIT_SELECT = 1;），接着送苹果（P0 = 0x00，位选数据），完了关闭我和张三的通信开关（BIT_SELECT = 0;）；之后打开我和李四的通信开关（SEG_SELECT = 1;），接着送桃子（P0 = 0x06，段选数据），再关闭我和李四的通信开关（SEG_SELECT = 0;）。看见了没，我一个人，在不同的时候给了别人不同的东西，这就是时分复用。

5.6 例说数码管

实验 1 数码管静态显示

什么是静态显示，这是相对于动态扫描来说的，没有一个明确的定义。要说定义就是读者看到数据就是数码管真实显示的数据，就是再慢、再快都是一样的数据。例如我们就取个时间点（仅仅一个点），读者若看到的数码管显示为：6666 6666，那么说明这个 8 个数码管确实都显示 6，而动态扫描中就不是了，具体到实例 10 中再做说明。

- **实验目的：**学会如何控制数码管静态显示

- **实验器材：**电脑、麦光开发板、Keil4 编译软件、ISP 烧录软件
- **实验原理：**看下面程序的解释
- **实验步骤：**
  - 编写程序，并生成 HEX 文件

```
1.  #include <reg52.h>
2.  #define uChar8 unsigned char
3.  #define uInt16 unsigned int
4.
5.  uChar8 code Disp_Tab[] =
6.  {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
7.
8.  #define DATA P0                         //P0 口宏定义
9.  sbit SEG_SELECT = P1^7;                 //段选定义
10. sbit BIT_SELECT = P1^6;                 //位选定义
11.
12. void DelayMS(uInt16 ValMS)
13. {
14.       uInt16 uiVal,ujVal;
15.       for(uiVal = 0; uiVal < ValMS; uiVal++)
16.             for(ujVal = 0; ujVal < 113; ujVal++);
17. }
18.
19. void main(void)
20. {
21.       uChar8 uiVal;
22.       BIT_SELECT = 1;                   //位选开
23.       DATA = 0x00;                      //送入位选数据
24.       BIT_SELECT = 0;                   //位选关
25.       while(1)
26.       {
27.             for(uiVal = 0;uiVal < 10;uiVal++)
28.             {
29.                   SEG_SELECT = 1;             //段选开
30.                   DATA = Disp_Tab[i];         //送入段选数据
31.                   SEG_SELECT = 0;             //段选关
32.                   DelayMS(500);               //延时
33.             }
34.       }
35. }
```

  - 程序分析

接下来简单分析一下程序，第 2~3 行就是起别名，便于以后操作，这里说明一点，C 语言真正起别名的不是#define，而是 typedef，具体详解请看上面的 C 语言基础部分；第 5~6 行定义了一个数组，总共十个元素，分别代表 0~9 这 10 个数字的编码，具体怎么编码来的，前面有说，要不啰嗦一下，例如 0 的编码，要亮 0，意味着 a、b、c、d、e、f 要亮，g、dp

灭，从而对应的二进制数为：0b1111 1100，但在单片机中高位在前，所以数据应为：0b0011 1111，这样就是第一个元素 0x3f，别的同理；第 22~24 则为送位选数据，先开门再送 0x00(8 个全部选中)，接着关门。接着送 10 次段选数据，也可以像点亮“1”的那样写，CTRL+C、CTRL+V10 次，只是有点累，所以用了 for 循环，循环 10 次，送 10 次数据，送的过程也是先开门，再送数据，后关门。第 32 行，延时主要是便于观察，读者可以任意修改，如果各位读者都有火眼金睛，那我说你随意，不延时都行。

实验 2 数码管动态扫描

先来解释一下动态，解释静态时说是相对于动态的，那究竟动态的参照物又是什么？其实这个没什么参考标准，前面说过，静态中，读者看到的就是数码管真实显示的数字，那难道动态扫描中看到的就不是数码管真实显示的数字呢，答案是：NO。

所谓动态扫描，实际上是轮流点亮数码管（某一个时刻内只有一个数码管是亮的），之后利用人眼的视觉暂留现象（也即余辉效应），当着 8 个数码管扫描的速度足够快时，给人一种感觉（仅仅是一种感觉哈，一点都不“真实”）是这 8 个数码管是同时亮了，这就是动态扫描显示的含义。例如要动态显示：01234567，显示过程就是先让第一个显示 0，过一会（小于某个时间，读者暂时现用 2ms 吧），接着让第二个显示 1，依次类推，让八个数码管分别显示 0~7，由于刷新的时间太快，给大家一种感觉是都在亮，事实上，看上去的这个时刻点只有一个数码管在显示。人是很聪明，但是眼睛的速度一直没有 CPU 快，因此聪明的人们利用这点做了好多文章。例如读者现在所用的电脑，或者大街上看到的 LED 显示屏等，都是利用人视觉暂留效应应用而生的，只是产生原理有别罢了。那么又会有一个新问题，两者之间的时间差多少为宜？这个问题放到下一讲再详细说明。接着来看看广为流传的动态扫描源代码。

```
1.  #include <reg52.h>
2.  #define uChar8 unsigned char
3.  #define uInt16 unsigned int
4.  #define DATA P0                                    //宏定义数据端口
5.  sbit SEG_SELECT = P1^7;                            //段选
6.  sbit BIT_SELECT = P1^6;                            //位选
7.  uChar8 code Bit_Tab[] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};     //位选数组
8.  uChar8 code Disp_Tab[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07};//0~7 数字的编码
9.  void DelayMS(uInt16 ValMS)
10. {    /* 两个 for 循环的魅力 */       }
11. void main(void)
12. {
13.      uInt16 uiVal;
14.      while(1)
15.      {
16.          for(uiVal = 0;uiVal < 8;uiVal++)
17.          {
18.              BIT_SELECT = 1;            //位选选通
19.              DATA = Bit_Tab[uiVal];     //送位选数据
20.              BIT_SELECT = 0;            //位选关闭
21.              SEG_SELECT = 1;            //段选选通
22.              DATA = Disp_Tab[uiVal];    //送段选数据
```

```
23.                 SEG _SELECT = 0;            //段选关闭
24.                 DelayMS(2);                 //延迟，就是两个数码管之间的时间差
25.             }
26.         }
27. }
```

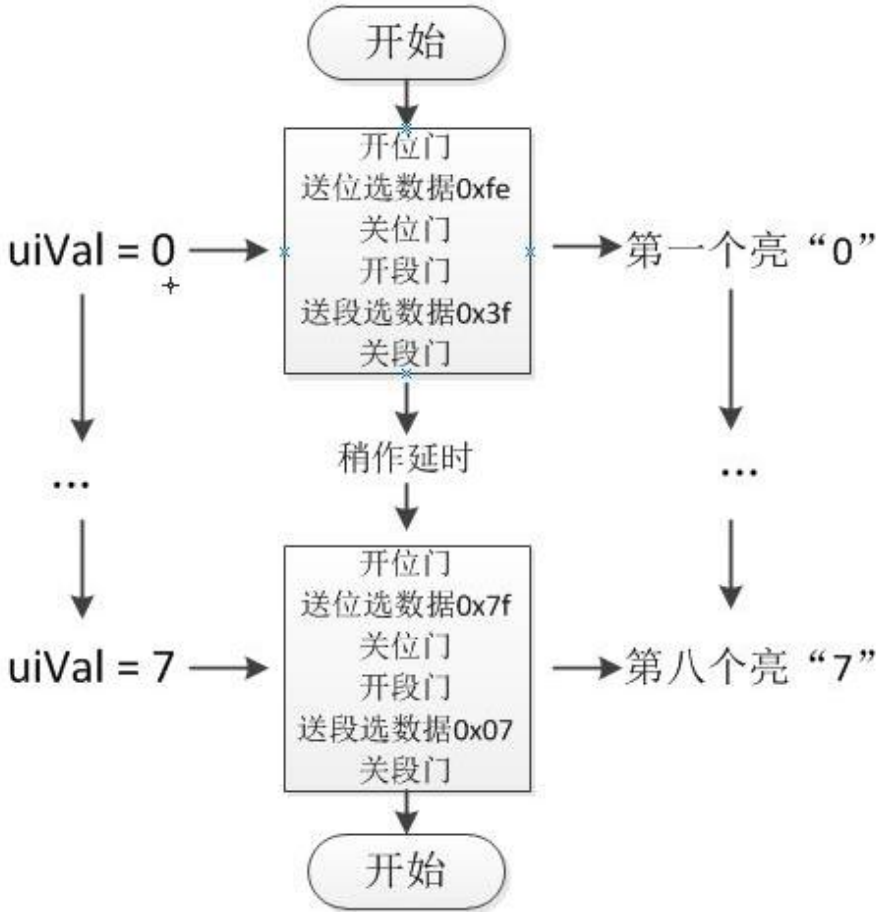

图 5-6　动态扫描流程图

同样残弈悟恩在这里先画一个“山寨版”的流程图，如图 5-6 所示。uiVal 等于 0 时，先开位门，再送位选数据（0xfe），关位门；先开段门，再送段选数据（0x3f），关段门。同理，uiVal 等于 1、...、7，接着依次送位选、段选数据，只是 uiVal 从 0 到 7 执行的比较快罢了，因而产生了如图 5-7 的效果。这个程序时广为流传的动态扫描代码，自己觉得该程序还是有一点问题，或许是由于手机不好还是残弈悟恩的拍摄水平太好（呵呵）的缘故，这里没有拍摄出“鬼影”的现象，什么是“鬼影”，就是不想让亮的那段隐隐约约的还在发亮，具体读者们可以自己做实验来验证，看是否有该现象，关于这个问题，后面学了定时器之后再做讲解，小伙伴们就先拭目以待吧。

图 5-7　数码管动态显示效果图