

上海沪生电子产品使用说明书

<http://www.husoon.com>

电话 : 021-61021969

网站 : <http://www.husoon.com>

第一章 Keil 工程文件的建立、设置与目标文件的获得

单片机开发中除必要的硬件外，同样离不开软件，我们写的汇编语言源程序要变为 CPU 可以执行的机器码有两种方法，一种是手工汇编，另一种是机器汇编，目前已极少使用手工汇编的方法了。机器汇编是通过汇编软件将源程序变为机器码，用于 80C51 单片机的汇编软件有早期的 A51，随着单片机开发技术的不断发展，从普遍使用汇编语言到逐渐使用高级语言开发，单片机的开发软件也在不断发展，Keil 软件是目前最流行开发 80C51 系列单片机的软件，这从近年来各仿真机厂商纷纷宣布全面支持 Keil 即可看出。Keil 提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（uVision）将这些部份组合在一起。运行 Keil 软件需要 Pentium 或以上的 CPU，16MB 或更多 RAM、20M 以上空闲的硬盘空间、WIN98、NT、WIN2000、WINXP 等操作系统。掌握这一软件的使用对于使用 80C51 系列单片机的爱好者来说是十分必要的，如果你使用 C 语言编程，那么 Keil 几乎就是你的不二之选（目前在国内你只能买到该软件、而你买的仿真机也很可能只支持该软件），即使不使用 C 语言而仅用汇编语言编程，其方便易用的集成环境、强大的软件仿真调试工具也会令你事半功倍。

我们将通过一些实例来学习 Keil 软件的使用，在这一部份我们将学习如何输入源程序，建立工程、对工程进行详细的设置，以及如何将源程序变为目标代码。图 1 所示电路图使用 89C51 单片机作为主芯片，这种单片机属于 80C51 系列，其内部有 4K 的 FLASH ROM，可以反复擦写，非常适于做实验。89C51 的 P1 引脚上接 8 个发光二极管，P3.2~P3.4 引脚上接 4 个按钮开关，我们的第一个任务是让接在 P1 引脚上的发光二极管依次循环点亮。

1.1 Keil 工程的建立

首先启动 Keil 软件的集成开发环境，这里假设读者已正确安装了该软件，可以从桌面上直接双击 uVision 的图标以启动该软件。

UVision 启动后，程序窗口的左边有一个工程管理窗口，该窗口有 3 个标签，分别是 Files、Regs、和 Books，这三个标签页分别显示当前项目的文件结构、CPU 的寄存器及部份特殊功能寄存器的值（调试时才出现）和所选 CPU 的附加说明文件，如果是第一次启动 Keil，那么这三个标签页全是空的。

1.1.1 源文件的建立

使用菜单“File->New”或者点击工具栏的新建文件按钮，即可在项目窗口的右侧打开一个新的文本编辑窗口，在该窗口中输入以下汇编语言源程序，例 1：

```

MOV    A, #0FEH
MAIN:  MOV    P1, A
        RL     A
        LCALL  DELAY
        AJMP  MAIN
DELAY:  MOV    R7, #255
D1:    MOV    R6, #255

```

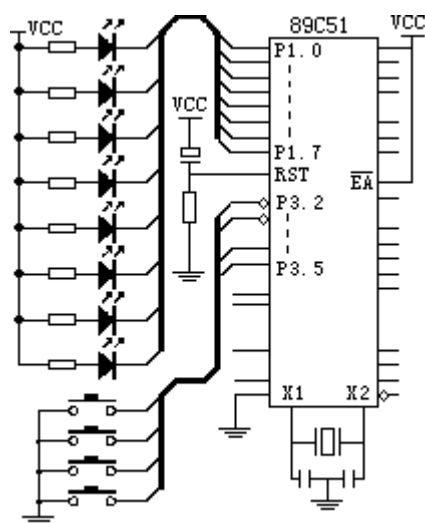


图 1-1 简单的键盘、显示板

```

DJNZ R6,$
DJNZ R7,D1
RET

```

END

保存该文件，注意必须加上扩展名（汇编语言源程序一般用 asm 或 a51 为扩展名），这里假定将文件保存为 exam1.asm。

需要说明的是，源文件就是一般的文本文件，不一定使用 Keil 软件编写，可以使用任意文本编辑器编写，而且，Keil 的编辑器对汉字的支持不好，建议使用 UltraEdit 之类的编辑软件进行源程序的输入。

1.1.2 建立工程文件

在项目开发中，并不是仅有一个源程序就行了，还要为这个项目选择 CPU（Keil 支持数百种 CPU，而这些 CPU 的特性并不完全相同），确定编译、汇编、连接的参数，指定调试的方式，有一些项目还会有多个文件组成等，为管理和使用方便，Keil 使用工程（Project）这一概念，将这些参数设置和所需的所有文件都加在一个工程中，只能对工程而不能对单一的源程序进行编译（汇编）和连接等操作，下面我们就一步一步地来建立工程。

点击“Project->New Project...”菜单，出现一个对话框，要求给将要建立的工程起一个

名字，你可以在编辑框中输入一个名字（设为 exam1），不需要扩展名。点击“保存”按钮，出现第二个对话框，如图 2 所示，这个对话框要求选择目标 CPU（即你所用芯片的型号），Keil 支持的 CPU 很多，我们选择 Atmel 公司的 89C51 芯片。点击 ATMEL 前面的“+”号，展开该层，点击其中的 89C51，然后再点击“确定”按钮，回到主界面，此时，在工程窗口的文件页中，出现了“Target 1”，前面有“+”号，点击“+”号展开，可以看到下一层的“Source Group1”，这时的工程还是一个空的工程，里面什么文件也没有，需要手动把刚才编写好的源程序加入，点击“Source

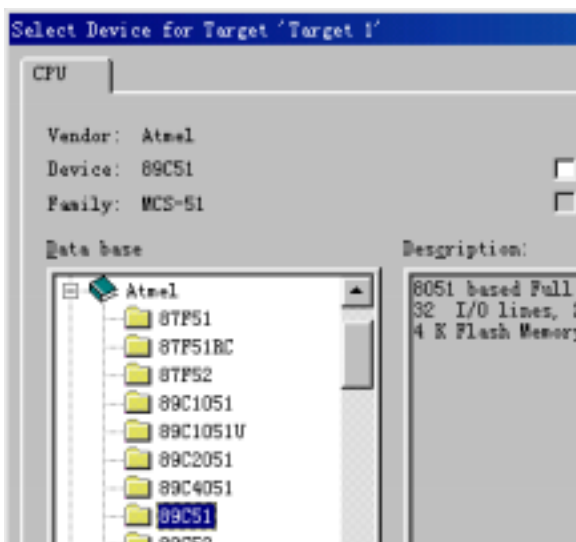


图 1-2 选择目标 CPU

Group1”使其反白显示，然后，点击鼠标右键，出现一个下拉菜单，如图 3 所示。选中其中的“Add file to Group”Source Group1”，出现一个对话框，要求寻找源文件，注意，该对话框下面的“文件类型”默认为 C source file(*.c)，也就是以 C 为扩展名的文件，而我们的文件是以 asm 为扩展名的，所以在列表框中找不到 exam1.asm，要将文件类型改掉，点击对话框中“文件类型”后的下拉列表，找到并选中“Asm Source File(*.a51,*.asm)”，这样，在列表框中就可以找到 exam1.asm 文件了。

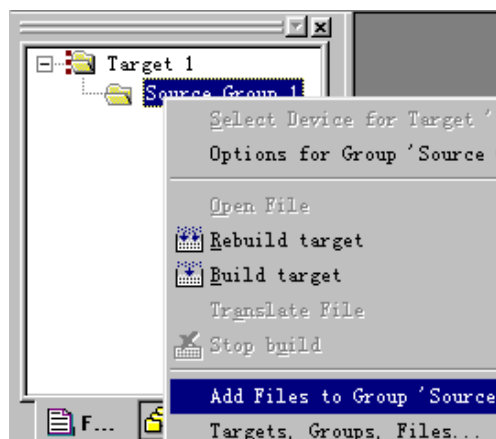


图 1-3 加入文件

双击 exam1.asm 文件，将文件加入项目，注意，在文件加入项目后，该对话框并不消失，等待继续加入其它文件，但初学时常会误认为操作没有成功而再次双击同一文件，这时会出现如图 4 所示的对话框，提示你所选文件已在列表中，此时应点击“确定”，返回前一对话框，然后点击“Close”即可返回主界面，返回后，点击“Source Group 1”前的加号，会发现 exam1.asm 文件已在其中。双击文件名，即打开该源程序。



图 1-4 重复加入文件的错误

1.2 工程的详细设置

工程建立好以后，还要对工程进行进一步的设置，以满足要求。

首先点击左边 Project 窗口的 Target 1，然后使用菜单“Project->Option for target 'target1'”即出现对工程设置的对话框，这个对话框可谓非常复杂，共有 8 个页面，要全部搞清可不容易，好在绝大部份设置项取默认值就行了。

设置对话框中的 Target 页面，如图 5 所示，Xtal 后面的数值是晶振频率值，默认值是所选目标 CPU 的最高可用频率值，对于我们所选的 AT89C51 而言是 24M，该数值与最终产生的目标代码无关，仅用于软件模拟调试时显示程序执行时间。正确设置该数值可使显示时间与实际所用时间一致，一般将其设置成与你的硬件所用晶振频率相同，如果没必要了解程序执行的时间，也可以不设，这里设置为 12。

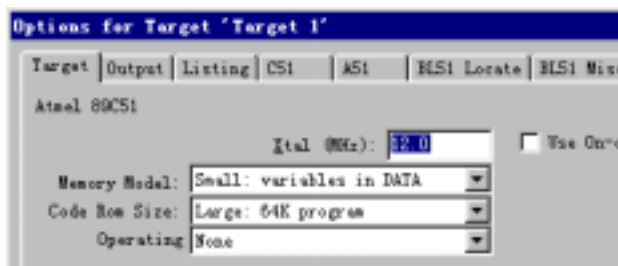


图 1-5 对目标进行设置

Memory Model 用于设置 RAM 使用情况，有三个选择项，Small 是所有变量都在单片机的内部 RAM 中；Compact 是可以使用一页外部扩展 RAM，而 Large 则是可以使用全部外部的扩展 RAM。Code Model 用于设置 ROM 空间的使用，同样也有三个选择项，即 Small 模式，只用低于 2K 的程序空间；Compact 模式，单个函数的代码量不能超过 2K，整个程序可以使用 64K 程序空间；Large 模式，可用全部 64K 空间。Use on-chip ROM 选择项，确认是否仅使用片内 ROM（注意：选中该项并不会影响最终生成的目标代码量）；Operating 项是操作系统选择，Keil 提供了两种操作系统：Rtx tiny 和 Rtx full，关于操作系统是另外一个很大的话题了，通常我们不使用任何操作系统，即使用该项的默认值：None（不使用任何操作系统）；Off Chip Code memory 用以确定系统扩展 ROM 的地址范围，Off Chip xData memory 组用于确定系统扩展 RAM 的地址范围，这些选择项必须根据所用硬件来决定，由于该例是单片应用，未进行任何扩展，所以均不重新选择，按默认值设置。

设置对话框中的 OutPut 页面，如图 6 所示，这里面也有多个选择项，其中 Creat Hex file 用于生成可执行代码文件（可以用编程器写入单片机芯片的 HEX 格式文件，文件的扩展名为.HEX），默认情况下该项未被选中，如果要写片做硬件实验，就必须选中该项，这一点是初学者易疏忽的，在此特别提醒注意。选中 Debug information 将会产生调试信息，这些信息用于调试，如果需要对程序进行调试，应当选中该项。Browse information 是产生浏览信息，该信息可以用菜单 view->Browse 来查看，这里取默认值。按钮“Select Folder for

objects”是用来选择最终的目标文件所在的文件夹，默认是与工程文件在同一个文件夹中。Name of Executable 用于指定最终生成的目标文件的名称，默认与工程的名字相同，这两项一般不需要更改。

工程设置对话框中的其它各页面与 C51 编译选项、A51 的汇编选项、BL51 连接器的连接选项等用法有关，这里均取默认值，不作任何修改。以下仅对一些有关页面中常用的选项作一个简单介绍。

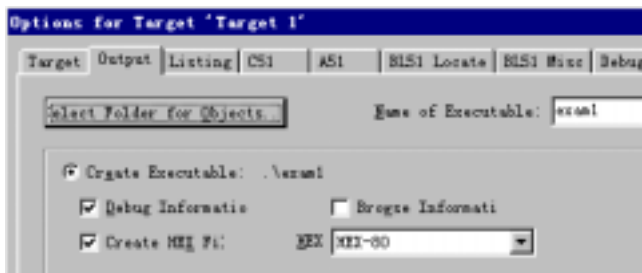


图 1-6 对输出进行控制

Listing 标签页用于调整生成的列表文件选项。在汇编或编译完成后将产生 (*.lst) 的列表文件，在连接完成后也将产生 (*.m51) 的列表文件，该页用于对列表文件的内容和形式进行细致的调节，其中比较常用的选项是“C Compile Listing”下的“Assamble Code”项，选中该项可以在列表文件中生成 C 语言源程序所对应的汇编代码。

C51 标签页用于对 Keil 的 C51 编译器的编译过程进行控制，其中比较常用的是“Code Optimization”组，如图 7 所示，该组中 Level 是优化等级，C51 在对源程序进行编译时，可以对代码多至 9 级优化，默认使用第 8 级，一般不必修改，如果在编译中出现一些问题，可以降低优化级别试一试。Emphasis 是选择编译优先方式，第一项是代码量优化（最终生成的代码量小）；第二项是速度优先（最终生成的代码速度快）；第三项是缺省。默认的是速度优先，可根据需要更改。

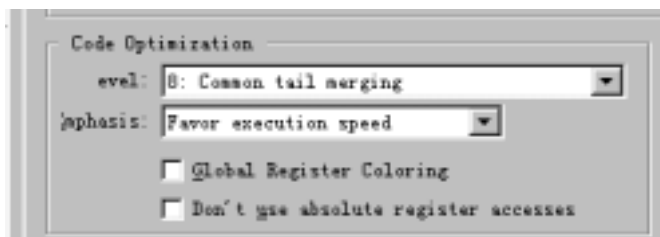


图 1-7 代码生成控制

设置完成后按确认返回主界面，工程文件建立、设置完毕。

1.3 编译、连接

在设置好工程后，即可进行编译、连接。选择菜单 Project->Build target 对当前工程进行连接，如果当前文件已修改，软件会先对该文件进行编译，然后再连接以产生目标代码；如果选择 Rebuild All target files 将会对当前工程中的所有文件重新进行编译然后再连接，确保最终生产的目标代码是最新的，而 Translate项则仅对该文件进行编译，不进行连接。



图 1-8 有关编译、连接、项目设置的工具条

以上操作也可以通过工具栏按钮直接进行。图 8 是有关编译、设置的工具栏按钮，从左到右分别是：编译、编译连接、全部重建、停止编译和对工程进

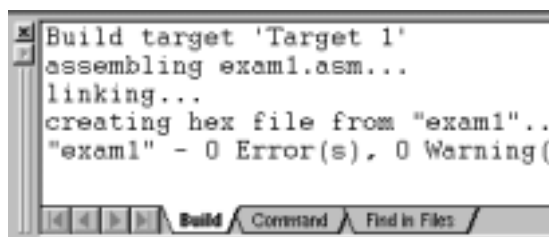


图 1-9 正确编译、连接之后的结果

行设置。

编译过程中的信息将出现在输出窗口中的 Build 页中，如果源程序中有语法错误，会有错误报告出现，双击该行，可以定位到出错的位置，对源程序反复修改之后，最终会得到如图 9 所示的结果，提示获得了名为 exam1.hex 的文件，该文件即可被编程器读入并写到芯片中，同时还产生了一些其它相关的文件，可被用于 Keil 的仿真与调试，这时可以进入下一步调试的工作。

第二章 Keil 的调试命令、在线汇编与断点设置

上一讲中我们学习了如何建立工程、汇编、连接工程，并获得目标代码，但是做到这一步仅代表你的源程序没有语法错误，至于源程序中存在着其它错误，必须通过调试才能发现并解决，事实上，除了极简单的程序以外，绝大部份的程序都要通过反复调试才能得到正确的结果，因此，调试是软件开发中重要的一个环节，这一讲将介绍常用的调试命令、利用在线汇编、各种设置断点进行程序调试的方法，并通过实例介绍这些方法的使用。

2.1 常用调试命令

在对工程成功地进行汇编、连接以后，按 Ctrl+F5 或者使用菜单 Debug->Start/Stop Debug Session 即可进入调试状态，Keil 内建了一个仿真 CPU 用来模拟执行程序，该仿真 CPU 功能强大，可以在没有硬件和仿真机的情况下进行程序的调试，下面将要学的就是该模拟调试功能。不过在学习之前必须明确，模拟毕竟只是模拟，与真实的硬件执行程序肯定还是有区别的，其中最明显的就是时序，软件模拟是不可能和真实的硬件具有相同的时序的，具体的表现就是程序执行的速度和每人使用的计算机有关，计算机性能越好，运行速度越快。

进入调试状态后，界面与编辑状态相比有明显的变化，Debug 菜单项中原来不能用的命令现在已可以使用了，工具栏会多出一个用于运行和调试的工具条，如图 1 所示，Debug 菜单上的大部份命令可以在此找到对应的快捷按钮，从左到右依次是复位、运行、暂停、单步、过程单步、执行完当前子程序、运行到当前行、下一状态、打开跟踪、观察跟踪、反汇编窗口、观察窗口、代码作用范围分析、1# 串行窗口、内存窗口、性能分析、工具按钮等命令。



图 2-1 调试工具条

学习程序调试，必须明确两个重要的概念，即单步执行与全速运行。全速执行

是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，并可以看到该段程序执行的总体效果，即最终结果正确还是错误，但如果程序有错，则难以确认错误出现在哪些程序行。单步执行是每次执行一行程序，执行完该行程序以后即停止，等待命令执行下一行程序，此时可以观察该行程序执行完以后得到的结果，是否与我们写该程序所想要得到的结果相同，借此可以找到程序中问题所在。程序调试中，这两种运行方式都要用到。

使用菜单 STEP 或相应的命令按钮或使用快捷键 F11 可以单步执行程序，使用菜单 STEP OVER 或功能键 F10 可以以过程单步形式执行命令，所谓过程单步，是指将汇编语言中的子程序或高级语言中的函数作为一个语句来全速执行。

按下 F11 键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向源程序的第一行，如图 2 所示。每按一次 F11，即执行该箭头所指程序行，然后箭头指向下一行，当



图 2-2 调试窗口

箭头指向 LCALL DELAY 行时，再次按下 F11，会发现，箭头指向了延时子程序 DELAY 的第一行。不断按 F11 键，即可逐步执行延时子程序。

通过单步执行程序，可以找出一些问题的所在，但是仅依靠单步执行来查错有时是困难的，或虽能查出错误但效率很低，为此必须辅之以其它的方法，如本例中的延时程序是通过将 D2: DJNZ R6,D2 这一行程序执行六万多次来达到延时的目的，如果用按 F11 六万多次的方法来执行完该程序行，显然不合适，为此，可以采取以下一些方法，第一，用鼠标在子程序的最后一行 (ret) 点一下，把光标定位于该行，然后用菜单 Debug->Run to Cursor line (执行到光标所在行)，即可全速执行完黄色箭头与光标之间的程序行。第二，在进入该子程序后，使用菜单 Debug->Step Out of Current Function(单步执行到该函数外)，使用该命令后，即全速执行完调试光标所在的子程序或子函数并指向主程序中的下一行程序 (这里是 JMP LOOP 行)。第三种方法，在开始调试的，按 F10 而非 F11，程序也将单步执行，不同的是，执行到 lcall delay 行时，按下 F10 键，调试光标不进入子程序的内部，而是全速执行完该子程序，然后直接指向下一行“JMP LOOP”。灵活应用这几种方法，可以大大提高查错的效率。

2.2 在线汇编

在进入 Keil 的调试环境以后，如果发现程序有错，可以直接对源程序进行修改，但是要修改后的代码起作用，必须先退出调试环境，重新进行编译、连接后再次进入调试，如果只是需要对某些程序行进行测试，或仅需对源程序进行临时的修改，这样的过程未免有些麻烦，为此 Keil 软件提供了在线汇编的能力，将光标定位于需要修改的程序行上，用菜单 Debug->Inline Assembly...即可出现如图 3 的对话框，在 Enter New 后面的编辑框内直接输入需更改的程序语句，输入完后键入回车将自动指向下一条语句，可以继续修改，如果不再需要修改，可以点击右上角的关闭按钮关闭窗口。

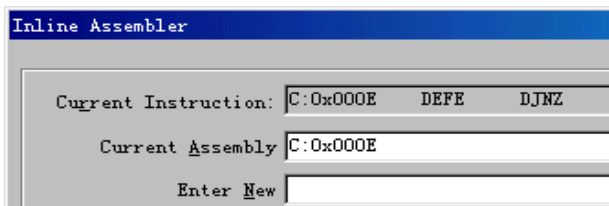


图 2-3 在线汇编窗口

2.3 断点设置

程序调试时，一些程序行必须满足一定的条件才能被执行到 (如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等)，这些条件往往是异步发生或难以预先设定的，这类问题使用单步执行的方法是很难调试的，这时就要使用到程序调试中的另一种非常重要的方法——断点设置。断点设置的方法有多种，常用的是在某一程序行设置断点，设置好断点后，可以全速运行程序，一旦执行到该程序行即停止，可在此观察有关变量值，以确定问题所在。在程序行设置/移除断点的方法是将光标定位于需要设置断点的程序行，使用菜单 Debug->Insert/Remove BreakPoint 设置或移除断点 (也可以用鼠标在该行双击实现同样的功能)；Debug->Enable/Disable Breakpoint 是开启或暂停光标所在行的断点功能；Debug->Disable All Breakpoint 暂停所有断点；Debug->Kill All BreakPoint 清除所有的断点设置。这些功能也可以用工具条上的快捷按钮进行设置。

除了在某程序行设置断点这一基本方法以外，Keil 软件还提供了多种设置断点的方法，按 Debug->Breakpoints...即出现一个对话框，该对话框用于对断点进行详细的设置，如图 4

所示。

图 4 中 Expression 后的编辑框内用于输入表达式，该表达式用于确定程序停止运行的条件，这里表达式的定义功能非常强大，涉及到 Keil 内置的一套调试语法，这里不作详细说明，仅举若干实例，希望读者可以举一反三。

- 1) 在 Expression 中键入 `a==0xf7`，再点击 Define 即定义了一个断点，注意，a 后有两个等号，意即相等。该表达式的含义是：如果 a 的值到达 0xf7 则停止程序运行。除使用相等符号之外，还可以使用 `>`, `>=`, `<`, `<=`, `!=`（不等于），`&`（两值按位与），`&&`（两值相与）等运算符。
- 2) 在 Expression 后中键入 Delay 再点击 Define，其含义是如果执行标号为 Delay 的行列中断。
- 3) 在 Expression 后中键入 Delay，按 Count 后的微调按钮，将值调到 3，其意义是当第三次执行到 Delay 时才停止程序运行。
- 4) 在 Expression 后键入 Delay，在 Command 后键入 `printf("SubRoutine 'Delay' has been Called\n")`，主程序每次调用 Delay 程序时并不停止运行，但会在输出窗口 Command 页输出一行字符，即 SubRoutine 'Delay' has been Called。其中“\n”的用途是回车换行，使窗口输出的字符整齐。
- 5) 设置断点前先在输出窗口的 Command 页中键入 `DEFINE int I`，然后在断点设置时同 4)，但是 Command 后键入 `printf("SubRoutine 'Delay' has been Called %d times\n", ++I)`，则主程序每次调用 Delay 时将会在 Command 窗口输出该字符及被调用的次数，如 SubRoutine 'Delay' has been Called 10 times。

对于使用 C 源程序语言的调试，表达式中可以直接使用变量名，但必须要注意，设置时只能使用全局变量名和调试箭头所指模块中的局部变量名。

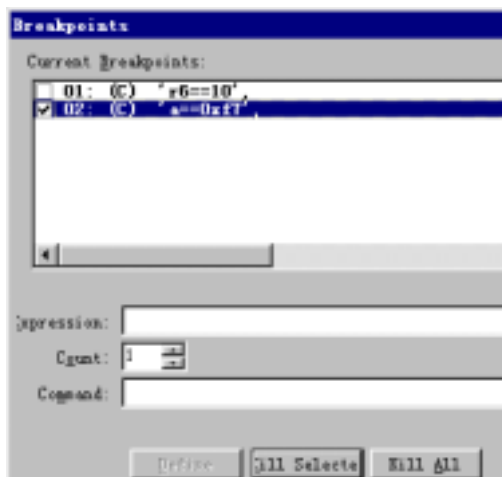


图 2-4 断点设置对话框

2.4 实例调试

例 2 是第一讲中出现过的流水灯程序，但其中的延时子程序书写方法略有不同，即“DJNZ R6,\$”指令改用标号的形式书写，效果是一样的，在这样书写时，很容易出的一个错误是将 D2 和 D1 混淆，即将“D2:DJNZ R6,D2”后面的 D2 误写成 D1，而将“DJNZ R7, D1”后的 D1 误写成 D2。下面我们就做一下这样的改动，然后重新编译，由于这样的改动并没有语法错误，所以，编译时不会报错。

例 2：流水灯程序

```
MOV    A, #0FEH
MAIN:  MOV    P1, A
        RL     A
        LCALL DELAY
        AJMP  MAIN
DELAY: MOV    R7, #255
D1:    MOV    R6, #255
```

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

进入调试状态后,按 F10 以过程单步的形式执行程序,当执行到 LCALL DELAY 行时,程序不能继续往下执行,同时发现调试工具条上的 Halt 按钮变成了红色,说明程序在此不断地执行着,而我们预期这一行程序也将执行完后停止,这个结果与预期不同,可以看出调用的子程序出了差错。为查明出错原因,按 Halt 按钮使程序停止执行,然后按 RST 按钮使程序复位,再次按下 F10 单步执行,但在执行到 LCALL DELAY 行时,改按 F11 键跟踪到子程序内部(如果按下 F11 键没有反应,请在源程序窗口中用鼠标点一下),单步执行程序,可以发现在执行到“D2:DJNZ R6,D1”行时,程序不断地从这一行转移到上一行,同时观察左侧的寄存器的值,会发现 R6 的值始终在 FFH 和 FEH 之间变化,不会减小,而我们的预期是 R6 的值不断减小,减到 0 后往下执行,因此这个结果与预期不符,通过这样的观察,不难发现问题是因为标号写错而产生的,发现问题即可以修改,为了验证即将进行的修改是否正确,可以先使用在线汇编功能测试一下。把光标定位于程序行 D2:DJNZ R6,D1,打开在线汇编的对话框,将程序改为 DJNZ R7,D2,回车后再键入 DJNZ R6,D1,然后关闭窗口,再进行调试,发现程序能够正确地执行了,这说明修改是正确的。注意,这时候的源程序并没有修改,此时应该退出调试程序,将源程序更改过来,并重新编译连接,以获得正确的目标代码。

第三章 Keil 程序调试窗口

上一讲中我们学习了几种常用的程序调试方法，这一讲中将介绍 Keil 提供各种窗口如输出窗口、观察窗口、存储器窗口、反汇编窗口、串行窗口等的用途，以及这些窗口的使用方法，并通过实例介绍这些窗口在调试中的使用。

3.1 程序调试时的常用窗口

Keil 软件在调试程序时提供了多个窗口，主要包括输出窗口 (Output Windows)、观察窗口 (Watch&Call Stack Windows)、存储器窗口 (Memory Window)、反汇编窗口 (Disassembly Window)、串行窗口 (Serial Window) 等。进入调试模式后，可以通过菜单 View 下的相应命令打开或关闭这些窗口。

图 1 是输出窗口、观察窗口和存储器窗口，各窗口的大小可以使用鼠标调整。进入调试程序后，输出窗口自动切换到 Command 页。该页用于输入调试命令和输出调试信息。对于初学者，可以暂不学习调试命令的使用方法。

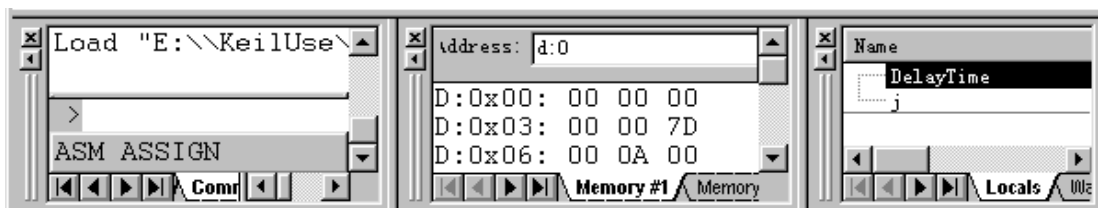


图 3-1 调试窗口 (命令窗口、存储器窗口、观察窗口)

3.1.1 存储器窗口

存储器窗口中可以显示系统中各种内存中的值，通过在 Address 后的编辑框内输入“字母：数字”即可显示相应内存值，其中字母可以是 C、D、I、X，分别代表代码存储空间、直接寻址的片内存储空间、间接寻址的片内存储空间、扩展的外部 RAM 空间，数字代表想要查看的地址。例如输入 D：0 即可观察到地址 0 开始的片内 RAM 单元值、键入 C：0 即可显示从 0 开始的 ROM 单元中的值，即查看程序的二进制代码。该窗口的显示值可以以各种形式显示，如十进制、十六进制、字符型等，改变显示方式的方法是点鼠标右键，在弹出的快捷菜单中选择，该菜单用分隔条分成三部份，其中第一部份与第二部份的三个选项为同一级别，选中第一部份的任一选项，内容将以整数形式显示，而选中第二部份的 Ascii 项则将以字符型式显示，选中 Float 项将相邻四字节组成的浮点数形式显示、选中 Double 项则将相邻 8 字节组成双精度形式显示。第一部份又有多个选择项，其中 Decimal 项是一个开关，如果选中该项，则窗口中的值将以十进制的形式显示，否则按默认的十六进制方式显示。Unsigned 和

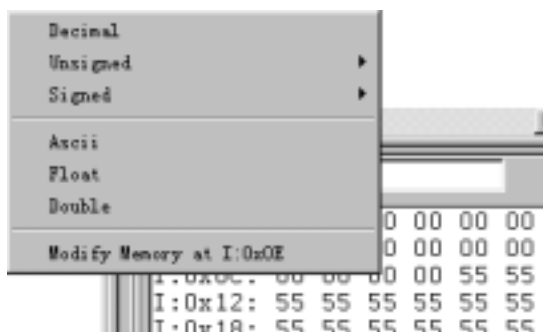


图 3-2 存储器数值各种方式显示选择

Signed 后分别有三个选项：Char、Int、Long，分别代表以单字节方式显示、将相邻双字节组成整型数方式显示、将相邻四字节组成长整型方式显示，而 Unsigned 和 Signed 则分别代表无符号形式和有符号形式，究竟从哪一个单元开始的相邻单元则与你的设置有关，以整型为例，如果你输入的是 I:0，那么 00H 和 01H 单元的内容将会组成一个整型数，而如果你输入的是 I:1，01H 和 02H 单元的内容全组成一个整型数，以此类推。有关数据格式与 C 语言规定相同，请参考 C 语言书籍，默认以无符号单字节方式显示。第三部份的 Modify Memory at X:xx 用于更改鼠标处的内存单元值，选中该项即出现如图 3 所示的对话框，可以在对话框内输入要修改的内容。

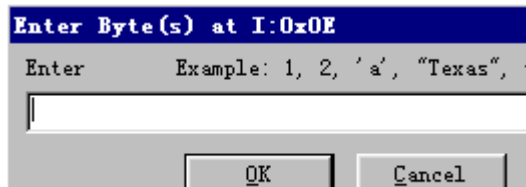


图 3-3 存储器的值的修改

3.1.2 工程窗口寄存器页

图 4 是工程窗口寄存器页的内容，寄存器页包括了当前的工作寄存器组和系统寄存器，系统寄存器组有一些是实际存在的寄存器如 A、B、DPTR、SP、PSW 等，有一些是实际中并不存在或虽然存在却不能对其操作的如 PC、Status 等。每当程序中执行到对某寄存器的操作时，该寄存器会以反色（蓝底白字）显示，用鼠标单击然后按下 F2 键，即可修改该值。

3.1.3 观察窗口

观察窗口是很重要的一个窗口，工程窗口中仅可以观察到工作寄存器和有限的寄存器如 A、B、DPTR 等，如果需要观察其它的寄存器的值或者在高级语言编程时需要直接观察变量，就要借助于观察窗口了。

其它窗口将在以下的实例中介绍。

一般情况下，我们仅在单步执行时才对变量的值的变化感兴趣，全速运行时，变量的值是不变的，只有在程序停下来之后，才会将这些值最新的变化反映出来，但是，在一些特殊场合下我们也可能需要在全速运行时观察变量的变化，此时可以点击 View->Periodic Window Update（周期更新窗口），确认该项处于被选中状态，即可在全速运行时动态地观察有关值的变化。但是，选中该项，将会使程序模拟执行的速度变慢。

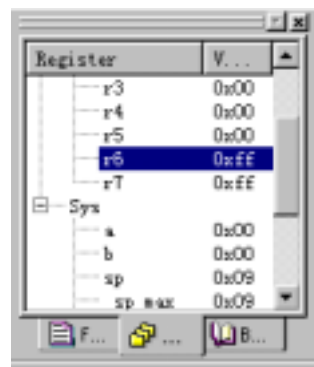


图 3-4 工程窗口寄存器

3.2 各种窗口在程序调试中的用途

以下通过一个高级语言程序来说明这些窗口的使用。例 2：

```
#include "reg51.h"
sbit P1_0=P1^0; //定义 P1.0
void mDelay(unsigned char DelayTime)
{
    unsigned int j=0;
    for(;DelayTime>0;DelayTime--)
        { for(j=0;j<125;j++) {} }
}
```

```

}
void main()
{
    unsigned int i;
    for(;;){
        mDelay(10); //延时 10 毫秒
        i++;
        if(i==10)
        {
            P1_0=!P1_0;
            i=0;
        }
    }
}

```

这个程序的工作过程是：不断调用延时程序，每次延时 10 毫秒，然后将变量 I 加 1，随后对变量 I 进行判断，如果 I 的值等于 10，那么将 P1.0 取反，并将 I 清 0，最终的执行效果是 P1.0 每 0.1S 取反一次。

输入源程序并以 exam2.c 为文件名存盘，建立名为 exam2 的项目，将 exam2.c 加入项目，编译、连接后按 Ctrl+F5 进入调试，按 F10 单步执行。注意观察窗口，其中有一个标签页为 Locals，这一页会自动显示当前模块中的变量名及变量值。可以看到窗口中有名为 I 的变量，其值随着执行的次数而逐渐加大，如果在执行到 mDelay(10)行时按 F11 跟踪到 mDelay 函数内部，该窗口的变量自动变为 DelayTime 和 j。另外两个标签页 Watch #1 和 Watch #2 可以加入自定义的观察变量，点击“type F2 to edit”然后再按 F2 即可输入变量，试着在 Watch #1 中输入 I，观察它的变化。在程序较复杂，变量很多的场合，这两个自定义观察窗口可以筛选出我们自己感兴趣的变量加以观察。观察窗口中变量的值不仅可以观察，还可以修改，以该程序为例，I 须加 10 次才能到 10，为快速验证是否可以正确执行到 P1_0=!P1_0 行，点击 I 后面的值，再按 F2，该值即可修改，将 I 的值改到 9，再次按 F10 单步执行，即可以很快执行到 P1_0=!P1_0 程序行。该窗口显示的变量值可以以十进制或十六进制形式显示，方法是在显示窗口点右键，在快捷菜单中选择如图 5 所示。

点击 View->Dissassembly Window 可以打开反汇编窗口，该窗口可以显示反汇编后的代码、源程序和相应反汇编代码的混合代码，可以在该窗口进行在线汇编、利用该窗口跟踪已找行的代码、在该窗口按汇编代码的方式单步执行，这也是一个重要的窗口。打开反汇编窗口，点击鼠标右键，出现快捷菜单，如图 6 所示，其中 Mixed Mode 是以混合方式显示，Assembly Mode 是以反汇编码方式显示。

程序调试中常使用设置断点然后全速运行的方式，在断点处可以获得各变量值，但却无法知道程序到达断点以前究竟执行了哪些代码，而这往往是需要了解的，为此，Keil 提供了跟踪功能，在运行程序之前打开调试工具条上的允许跟踪代码开关，然后全速运行程序，当程序停止运行后，点击查看跟踪代码按钮，自动切换到反汇编窗口，如图 6 所示，其中前面标有“-”号的行就是中断以前执行的代码，可以按窗口边的上卷按钮向上翻查看代码执行记录。

利用工程窗口可以观察程序执行的时间，下面我们观察一下该例中延时程序的延时时

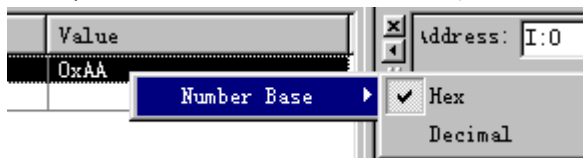


图 3-5 设定观察窗的显示方式

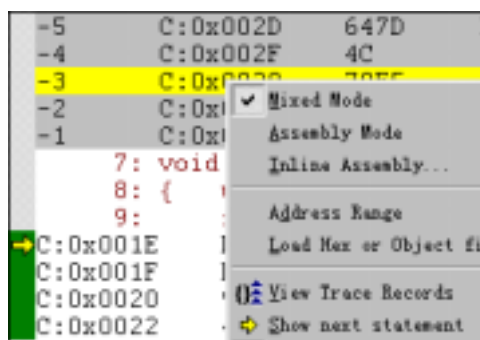


图 3-6 反汇编窗口

间是否满足我们的要求,即是否确实延时 10 毫秒,展开工程窗口 Regs 页中的 Sys 目录树,其中的 Sec 项记录了从程序开始执行到当前程序流逝的秒数。点击 RST 按钮以复位程序,Sec 的值回零,按下 F10 键,程序窗口中的黄色箭头指向 mDelay(10)行,此时,记录下 Sec 值为 0.00038900,然后再按 F10 执行完该段程序,再次查看 Sec 的值为 0.01051200,两者相减大约是 0.01 秒,所以延时时间大致是正确的。读者可以试着将延时程序中的 unsigned int 改为 unsigned char 试试看时间是否仍正确。注意,使用这一功能的前提是在项目设置中正确设置晶振的数值。

Keil 提供了串行窗口,我们可以直接在串行窗口中键入字符,该字符虽不会被显示出来,但却能传递到仿真 CPU 中,如果仿真 CPU 通过串行口发送字符,那么这些字符会在串行窗口显示出来,用该窗口可以在没有硬件的情况下用键盘模拟串口通讯。下面通过一个例子说明 Keil 串行窗口的应用。该程序实现一个行编辑功能,每键入一个字母,会立即回显到窗口中。编程的方法是通过检测 RI 是否等于 1 来判断串行口是否有字符输入,如果有字符输入,则将其送到 SBUF,这个字符就会在串行窗口中显示出来。其中 ser_init 是串行口初始化程序,要使用串行口,必须首先对串行口进行初始化。例 3:

```

MOV    SP,#5FH ;堆栈初始化
CALL   SER_INIT ;串行口初始化
LOOP:
JBC    RI,NEXT ;如果串口接收到字符,转
JMP    LOOP   ;否则等待接收字符
NEXT:
MOV    A,SBUF ;从 SBUF 中取字符
MOV    SBUF,A ;回送到发送 SBUF 中
SEND:
JBC    TI,LOOP ;发送完成,转 LOOP
JMP    SEND   ;否则等待发送完
SER_INIT:
;中断初始化
MOV    SCON,#50H
ORL    TMOD,#20H
ORL    PCON,#80H
MOV    TH1,#0FDH ;设定波特率
SETB   TR1 ;定时器 1 开始运行
SETB   REN ;允许接收
SETB   SM2
RET
END

```

输入源程序,并建立项目,正确编译、连接,进入调试后,全速运行,点击串行窗口 1 按钮,即在原源程序窗口位置出现一个空白窗口,击键,相应的字母就会出现在该窗口中。在窗口中击鼠标右键,出现一个弹出式菜单,选择“Ascii Mode”即以 Ascii 码的方式显示接收到的数据;选择“Hex Mode”以十六进制码方式显示接收到的数据;选择“Clear Window”可以清除窗口中显示的内容。

由于部份 CPU 具有双串口,故 Keil 提供了两个串行窗口,我们选用的 89C51 芯片只有一个串行口,所以 Serial 2 串行窗口不起作用。

小技巧:凡是鼠标单击然后按 F2 的地方都可以用鼠标连续单击两次(注意:不是双击)来替代。

第四章 Keil 的辅助工具和部份高级技巧

在前面的几讲中我们介绍了工程的建立方法，常用的调试方法，除此之外，Keil 还提供了一些辅助工具如外围接口、性能分析、变量来源分析、代码作用分析等，帮助我们了解解的性能、查找程序中的隐藏错误，快速查看程序变量名信息等，这一讲中将对这些工具作一介绍，另外还将介绍 Keil 的部份高级调试技巧。

4.1 辅助工具

这部份功能并不是直接用来进行程序调试的，但可以帮助我们进行程序的调试、程序性能的分析，同样是一些很有用的工具。

4.1.1 外围接口

为了能够比较直观地了解单片机中定时器、中断、并行端口、串行端口等常用外设的使用情况，Keil 提供了一些外围接口对话框，通过 Peripherals 菜单选择，该菜单的下拉菜单内容与你建立项目时所选的 CPU 有关，如果是选择的 89C51 这一类“标准”的 51 机，那么将会有 Interrupt(中断)、I/O Ports(并行 I/O 口)、Serial(串行口)、Timer(定时/计数器)这四个外围设备菜单。打开这些对话框，列出了外围设备的当前使用情况，各标志位的情况等，可以在这些对话框中直观地观察和更改各外围设备的运行情况。

下面我们通过一个简单例子看一看并行端口的外围设备对话框的使用。例 4：

```
MOV    A,#0FEH
LOOP:  MOV    P1,A
       RL     A
       CALL  DELAY ;延时 100 毫秒
       JMP   LOOP
```

其中延时 100 毫秒的子程序请自行编写。

编译、连接进入调试后，点击 Peripherals->I/O-Ports->Port 1 打开，如图 1 所示，全速运行，可以看到代表各位的勾在不断变化（如果看不到变化，请点击 View->Periodic Window Update），这样可以形象地看出程序执行的结果。

注：如果你看到的变化极快，甚至看不太清楚，那么说明你的计算机性能好，模拟执行的速度快，你可以试着将加长延时程序的时间以放慢速度。模拟运行速度与实际运行的速度无法相同是软件模拟的一个固有弱点。

点击 Peripherals->I/O-Ports->Timer0 即出现图 2 所示定时/计数器 0 的外围接口界面，可以直接选择 Mode 组

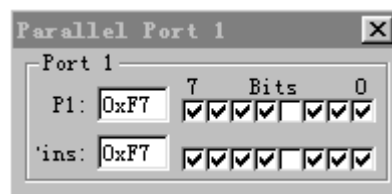


图 4-1 外围设备之并行端口



图 4-2 外围设备之定时器

中的下拉列表以确定定时/计数工作方式，0-3 四种工作方式，设定定时初值等，点击选中 TR0，status 后的 stop 就变成了 run，如果全速运行程序，此时 th0,tl0 后的值也快速地开始变化（同样要求 Periodic Window Update 处于选中状态），直观地演示了定时/计数器的工作情况（当然，由于你的程序未对此写任何代码，所以程序不会对此定时/计数器的工作进行处理）。

4.1.2 性能分析

Keil 提供了一个性能分析工具，利用该工具，我们可以了解程序中哪些部份的执行时间最长，调用次数最多，从而了解影响整个程序中执行速度的瓶颈。下面通过一个实例来看一看这个工具如何使用，例 5：

```
#include "reg51.h"
sbit P1_0=P1^0; //定义 P1.0
void mDelay(unsigned char DelayTime)
{
    unsigned int j=0;
    for(;DelayTime>0;DelayTime--)
    {
        for(j=0;j<125;j++)
        {
            ;
        }
    }
}
void mDelay1(unsigned char DelayTime)
{
    unsigned int j=0;
    for(;DelayTime>0;DelayTime--)
    {
        ;
    }
}

void main()
{
    unsigned int i;
    for(;;){
        mDelay(10); //延时 10 毫秒
        i++;
        if(i==10)
        {
            P1_0=!P1_0;
            i=0;
            mDelay1(10);}
    }
}
```

编译连接。进入调试状态后使用菜单 View->Performance Analyzer Window，打开性能分析对话框，进入该对话框后，只有一项 unspecified，点鼠标右键，在快捷菜单中选择 Setup PA 即打开性能分析设置对话框，对于 C 语言程序，该对话框右侧的“Function Symbol”下的列表框给出函数符号，双击某一符号，该符号即出现在 Define Performance Analyzer 下的编辑框中，每输入一个符号名字，点击 Define 按钮，即将该函数加入其上的分析列表框。对于汇编语言源程序，Function Symbol 下的列表框中不会出现子程序名，可以直接在编辑框中输入子程序名，点击 Close 关闭窗口，回到性能分析窗口，此时窗口共有 4 个选项。全速执行程序，可以看到 mDelay 和 mDelay1 后出现一个蓝色指示条，配合上面的标尺可以直观地看出每个函数占整个执行时间的比例，点击相应的函数名，可以在该窗口的状态栏看到更详细的数据，其中各项的含义如下：

Min：该段程序执行所需的最短时间；Max：该段程序执行所需的最长时间；Avg：该段程序执行所花平均时间；Total：该段程序到目前为目总共执行的时间；%：占整个执行时间的百分比；count：被调用的次数。

本程序中，函数 mDelay 和 mDelay1 每次被调用都花费同样的时间，看不出 Min、Max、和 Avg 的意义，实际上，由于条件的变化，某些函数执行的时间不一定是一个固定的值，借助于这些信息，可以对程序有更详细的了解。下面将 mDelay1 函数略作修改作一演示。

```
void mDelay1(unsigned char DelayTime)
{
    static unsigned char k;
    unsigned int j=0;
    for(;DelayTime>0;DelayTime--)
    {
        for(j<k;j++)
        {
            ;
        }
    }
}
```

```
    } k++; }
```

程序中定义了一个静态变量 K ,每次调用该变量加 1 ,而 j 的循环条件与 k 的大小有关 ,这使每次执行该程序所花的时间不一样。编译、执行该程序 ,再次观察性能分析窗口 ,可以看出 Min、Max、Avg 的意义。

4.1.3 变量来源浏览

该窗口用于观察程序中变量名的有关信息 ,如该变量名在那一个函数中被定义、在哪里被调用 ,共出现多少次等。在 Source Browse 窗口中提供了完善的管理方法 ,如过滤器可以分门别类地列出各种类别的变量名 ,可以对这些变量按 Class(组) Type(类型) Space (所在空间) Use (调用次数) 排序 ,点击变量名 ,可以在窗口的右侧看到该变量名的更详细的信息。

4.1.4 代码作用范围分析

在你写的程序中 ,有些代码可能永远不会被执行到 (这是无效的代码) ,也有一些代码必须在满足一定条件后才能被执行到 ,借助于代码范围分析工具 ,可以快速地了解代码的执行情况。

进入调试后 ,全速运行 ,然后按停止按钮 ,停下来后 ,可以看到在源程序的左列有三种颜色 ,灰、淡灰和绿 ,其中淡灰所指的行并不是可执行代码 ,如变量或函数定义、注释行等等 ,而灰色行是可执行但从未执行过的代码 ,而绿色则是已执行过的程序行。使用调试工具条上的 Code Coverage Window 可打开代码作用范围分析的对话框 ,里面有各个模块代码执行情况的更详细的分析。如果你发现全速运行后有一些未被执行到的代码 ,那么就要仔细分析 ,这些代码究竟是无效的代码还是因为条件没有满足而没有被执行到。

4.2 部份高级调试技巧

Keil 内置了一套调试语言 ,很多高级调试技巧与此有关 ,但是全面学习这套语言并不现实 ,这不是这么几期连载可以胜任的 ,这里仅介绍部份较为实用的功能 ,如要获得更详细的信息 ,请参考 Keil 自带的帮助文件 GS51.PDF。

4.2.1 串行窗口与实际硬件相连

Keil 的串行窗口除可以模拟串行口的输入和输出功能外还可以与 PC 机上实际的串口相连 ,接受串口输入的内容 ,并将输出送到串口。这需要在 Keil 中进行设置。方法是首先在输出窗口的 Command 页用 MODE 命令设置串口的工作方式 ,然后用 ASSIGN 命令将串行窗口与实际串口相关联 ,下面我们通过一个实例来说明如何操作。例 6 :

```

ORG    0000H
    JMP START
    ORG    3+4*8    ;串行中断入口
    JMP SER_INT
START:
    MOV    SP,#5FH    ;堆栈初始化
    CALL  SER_INIT ;串行口初始化 A
    SETB   EA    ;
    SETB   ES    ;
    JMP $    ;主程序到此结束
SER_INT:
    JBC    RI,NEXT ;如果串口接收到字符，转
    JMP    SEND    ;否则转发送处理
NEXT:
    MOV    A,SBUF    ;从 SBUF 中取字符
    MOV    SBUF,A    ;回送到发送 SBUF 中
    JMP    OVER
SEND:
    clr    ti
OVER:
    reti

SER_INIT:                ;中断初始化
    MOV    SCON,#50H
    ORL    TMOD,#20H
    ORL    PCON,#80H
    MOV    TH1,#0FDH ;设定波特率
    SETB   TR1    ;定时器 1 开始运行
    SETB   REN    ;允许接收
    SETB   SM2
    RET
END

```

这个程序使用了中断方式编写串行口输入/输出程序，它的功能是将接串行口收到的字符回送，即再通过串行口发送出去。

正确输入源文件、建立工程、编译连接没有错后，可进行调试，使用 Keil 自带的串行窗口测试功能是否正确，如果正确，可以进行下一步的联机试验。

为简单实用，我们不借助于其它的硬件，而是让 PC 机上的两个串口互换数据，即 COM1 发送 COM2 接收，而 COM2 发送则由 COM1 接收，为此，需要做一根连接线将这两个串口连起来，做法很简单，找两个可以插入 PC 机串口的 DIN9 插座（母），然后用一根 3 芯线将它们连起来，连线的的方法是：

```

2——3
3——2
5——5

```

接好线把两个插头分别插入 PC 机上的串口 1 与串口 2。找一个 PC 机上的串口终端调

试软件，如串口精灵之类，运行该软件，设置好串口参数，其中串口选择 2，串口参数设置为：

19200, n, 8, 1 其含义是波特率为 19200，无奇偶校验，8 位数据，1 位停止位。

在 Keil 调试窗口的 command 页中输入：

```
>mode com1 19200,0,8,1
```

```
>assign com1 <sin>sout
```

注意两行最前面的“>”是提示符，不要输入，第二行中的“<”和“>”即“小于”和“大于”符号，中间的是字母“s”和“input”的前两个字母，最后是字母“s”和“output”的前三个字母。

第一行命令定义串口 1 的波特率为 19200，无奇偶校验，8 位数据，1 位停止位。第二行是将串口 1 (com1) 分配给串行窗口。

全速运行程序，然后切换串口精灵，开始发送，会看到发送后的数据会立即回显到窗口中，说明已接收到了发送过来的数据。切换到 uVison，查看串行窗口 1，会看到这里的确接收到了串口精灵送来的内容。

4.2.2 从端口送入信号

程序调试中如果有信号输入，比如数据采集类程序，需要从外界获得数据，由于 Keil 的调试完全是一个软件调试工具，没有硬件与之相连，所以不可能直接获得数据，为此必须采用一些替代的方法，例如，某电路用 P1 口作为数据采集口，那么可以使用的一种方法是利用外围接口，打开 PORT 1，用鼠标在点击相应端口位，使其变为高电平或低电平，就能输入数据。显然，这种方法对于要输入数据而不是作位处理来说太麻烦了，另一种方法是直接在 command 页输入 port1=数值，以下是一个小小的验证程序。例 7：

```
LOOP:  MOV    A,P1
        JZ     NEXT
        MOV    R0,#55H
        JMP    LOOP
NEXT:  MOV    R0,#0AAH
        JMP    LOOP
        END
```

该程序从 P1 口获得数据，如果 P1 口的值是 0，那么就让 R0 的值为 0AAH，否则让 R0 的值为 55H。输入源程序并建立工程，进入调试后，在观察窗口加入 R0，然后全速运行程序，注意确保 View->Periodic Window Update 处于选中状态，然后在 Command 后输入 PORT1=0 回车后可以发现观察窗口中的 R0 的值变成了 0AAH，然后再输入 PORT1=1 或其它非零值，则 R0 的值会变为 55H。

同样的道理，可以用 port0、port2、port3 分别向端口 0、2、3 输入信号。

4.2.3 直接更改内存值

在程序运行中，另一种输入数据的方法是直接更改相应的内存单元的值，例如，某数据采集程序，使用 30H 和 31H 作为存储单元，采入的数据由这两个单元保存，那么我们更改了 30H 和 31H 单元的值就相当于这个数据采集程序采集到了数据，这可以在内存窗口中直接修改（参考上一讲），也可以通过命令进行修改，命令的形式是：_WBYTE(地址,数据)，其中地址是指待写入内存单元的地址，而数据则是待写入该地址的数据。例如 _WBYTE(0x30,11)会将值 11 写入内存地址十六进制 30H 单元中。

第五章 基于 Keil 的实验仿真板的使用

前面介绍了 Keil 软件的使用，从中我们可以看到 Keil 的强大功能，不过，对于初学者来说，还有些不直观，调试过程中看到的是一些数值，并没有看到这些数值所引起的外围电路的变化，例如数码管点亮、发光管发光等。为了让初学者更好地入门，笔者利用 Keil 提供的 AGSI 接口开发了两块仿真实验板。

这两块仿真板将枯燥无味的数字用形象的图形表达出来，可以使初学者在没有硬件时就能感受到真实的学习环境，降低单片机的入门门槛。图 1 是键盘、LED 显示实验仿真板的图，从图中可以看出，该板比较简单，有在 P1 口接有 8 个发光二极管，在 P3 口接有 4 个按钮，图的右边给出了原理图。

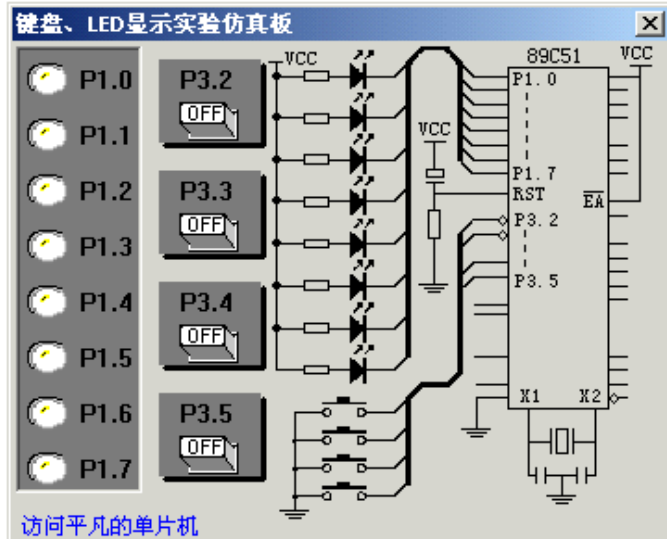


图 5-1 键盘、LED 显示实验仿真板

图 2 是另一个较为复杂的实验仿真板。在该板上有 8 个数码管，16 个按键（接成 4*4 的矩阵式），另外还有 P1 口接的 8 个发光管，两个外部中断按钮，一个带有计数器的脉冲发生器等资源，显然，这块板可以完成更多的实验。

5.1 实验仿真板的安装

这两块仿真实验板实际上是两个 dll 文件，名称分别是 ledkey.dll 和 simboard.dll，安装时只要根据需要将这两个或某一个文件拷贝到 keil 软件的 c51\bin 文件夹中即可。

5.2 实验仿真板的使用

要使用仿真板，必须对工程进行设置，设置的方法是点击 Project->Option for Target 'Target1' 打开对话框，然后选中 Debug 标签页，在 Dialog :Parameter: 后的编辑框中输入 -d 文件名。例如要用



图 5-2 单片机实验仿真板

ledkey.dll（即第一块仿真板）进行调试，就输入 -dledkey，如图 3 所示，输入完毕后点击确定退出。编译、连接完成后按 CTRL+F5 进入调试，此时，点击菜单 Peripherals，即会多出一项“键盘 LED 仿真板（K）”，选中该项，即会出现如图 1 的界面，同样，在设置时如果

输入-dsimboard 则能够调出如图 2 的界面。

第一块仿真板的硬件电路很简单，电路图已在板上，第二块板实现的功能稍复杂，其键盘和数码管显示管部份的电路原理图如图 4 所示。下表给出了常用字形码，读者也可以根据图中的接线自行写出其它如 A、B、C、D、E、F 等的字形码。除了键盘和数码管以外，P1 口同样也接有 8 个发光二极管，连接方式与图 1 相同，脉冲发生器是接入 T0 即 P3.4 引脚。

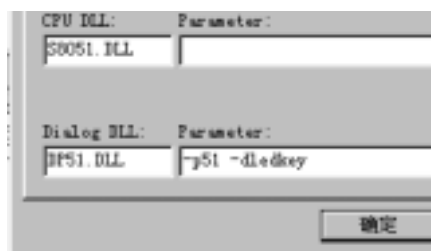


图 5-3 实验仿真板的设置

0c0h	0f9h	0a4h	0b0h	99h	92h	82h	0f8h	80h	90h	0FFH
0	1	2	3	4	5	6	7	8	9	消隐

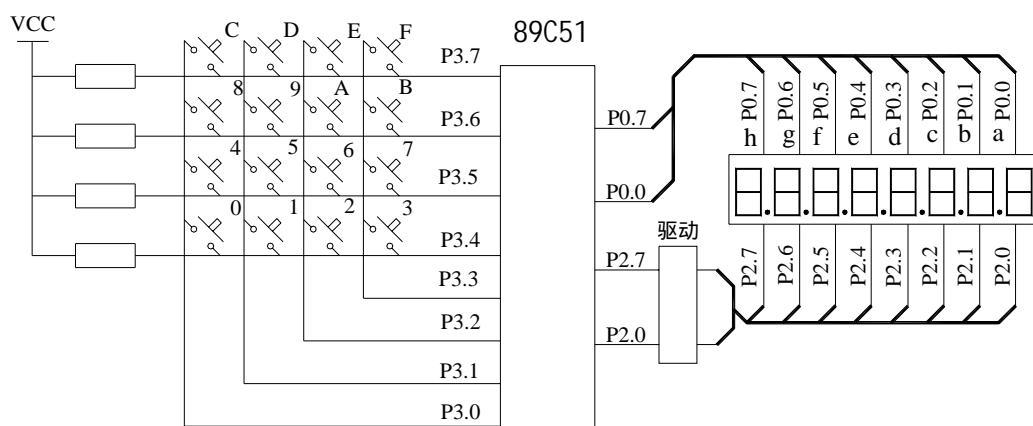


图 5-4 实验仿真板 2 数码管和键盘部份的电路图

5.3 实例调试

以下以一个稍复杂的程序为例，说明键盘、LED 显示实验仿真板的使用。该程序实现的是可控流水灯，接 P3.2 的键为开始键，按此键则灯开始流动（由上而下），接 P3.3 的键为停止键，按此键则停止流动，所有灯暗，接 P3.4 的键为向上键，按此键则灯由上向下流动，接 P3.5 的键为向下键，按此键则灯由下向上流动。

例 8：

```

UpDown    BIT 00H ;上下行标志
StartEnd  BIT 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

```

```

    CLRStartEnd ;启动时处于停止状态
    MOV    LAMPCODE,#01H;单灯流动的代码
LOOP:  ACALL KEY    ;调用键盘程序
    JNB F0,LNEXT   ;若无键按下,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
;延时程序,键盘处理中调用
DELAY: MOV    R7,#100
D1:  MOV    R6,#100
    DJNZ   R6,$
    DJNZ   R7,D1
    RET
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
    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:
    CLRStartEnd ;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:
    CLRUpDown      ;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,#00111100B ;将 P3 口的接有键的四位置 1
    MOV   A,P3 ;取 P3 的值
    ORL   A,#11000011B ;将其余 4 位置 1
    CPL  A ;取反
    JZ   K_RET ;如果为 0 则一定无键按下
    CALL DELAY ;否则延时去键抖
    ORL   P3,#00111100B
    MOV   A,P3
    ORL   A,#11000011B

```

```

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
;以下的代码是可以被注释掉的,如果去掉注释,就具有判断键是否释放的功能,否则
没有
K_RET: ;ORL P3,#00111100B ;此处循环等待键的释放
;MOV A,P3
;ORL A,#11000011B
;CPL A
;JZ K_RET1 ;读取的数据取反后为 0 说明键释放了
;AJMP K_RET
;K_RET1:CALL DELAY ;消除后沿抖动
RET
D500MS: ;流水灯的延迟时间
MOV R7,#255
D51: MOV R6,#255
DJNZ R6,$
DJNZ R7,D51
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 ;实际就是左移位而已
MOV LAMPCODE,A
MOV P1,A
LCALL D500MS
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

```

将程序输入并建立工程文件,设置工程文件,在 Debug 标签页中加入“-dledkey”,汇编、连接文件,按 Ctrl+F5 开始调试,打开仿真板,使用 F5 功能键全速运行,可以看到所

有灯均不亮，点击最上面的按钮，立即会看到灯流动起来了，点击第二个按钮，灯将停止流动，再次点击第一个按钮，使灯流动起来，点击第三个按钮，可以发现灯流动的方向变了，点击第四个按钮，灯的流动方向又变回来了。如果没有出现所描述的现象，可以使用单步、过程单步等调试手段进行调试，在进行调试时实验仿真板会随时显示出当前的情况，是不是非常的直观和方便呢？

下面的一个例子是关于第二块实验仿真板的，演示点亮 8 位数码管。例 9：

```

ORG    0000h
JMP    MAIN
ORG    30H
MAIN:
MOV    SP,#5FH
MOV    R1,#08H
MOV    R0,#58H    ;显示缓冲区首地址
MOV    A,#2
INIT:
MOV    @R0,A      ;初始化显示缓冲区
INC    A
INC    R0
DJNZ   R1,INIT    ;将 0-7 送显示缓冲区
LOOP:
CALL   DISPLAY
JMP    LOOP
;主程序到此结束
DISPLAY:
MOV    R0,#7FH    ;列选择
MOV    R7,#08H    ;共有 8 个字符
MOV    R1,#58H    ;显示缓冲区首地址
AGAIN:
MOV    A,@R1
MOV    DPTR,#DISPTABLE
MOVC   A,@A+DPTR
MOV    P0,A
MOV    P2,R0
MOV    A,R0
RR     A
MOV    R0,A
INC    R1
DJNZ   R7,AGAIN
RET
DISPTABLE: DB 0c0h,0f9h,0a4h,0b0h,99h,92h,82h,0f8h,80h,90h,0FFH    ;字形码表
END

```

这一程序内部 RAM 中 58H 到 5FH 被当成是显示缓冲区，主程序中用 2-9 填充该显示区，然后调用显示程序显示 2-9。这里是用最最简单的逐位显示的方式编写的显示程序。最后介绍一个小技巧，将鼠标移入按钮区域，按下左键，按钮显示被按下，不要放

开鼠标左键，将光标移出按钮区域，松开左键，可以看到，按钮仍处于按下状态，利用这一功能，在需要 I/O 口长期处于低电平时，你就不必一直用手按着鼠标的左键啦。