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，但是做到这一步仅仅代表你的源程序没有语法错误，至于源程序中存在着的其它错误，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。

一、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各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，界面与编缉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 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，是否与我们写该行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。

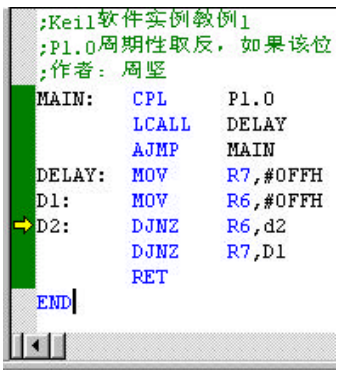

图 2 调试窗口

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。

二、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，但是要使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 Assambly…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。

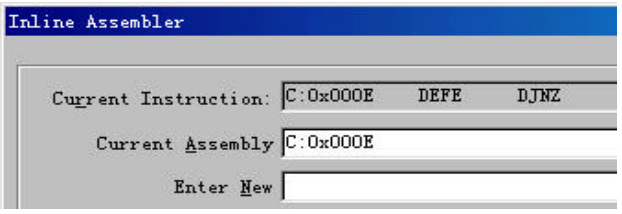

图 3 在线汇编窗口

三、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 Experssion 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）等运算符号。

2）在 Experssion 后中键入 Delay 再点击 Define，其含义是如果执行标号为 Delay 的行则中断。

3）在 Experssion 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 Experssion 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。

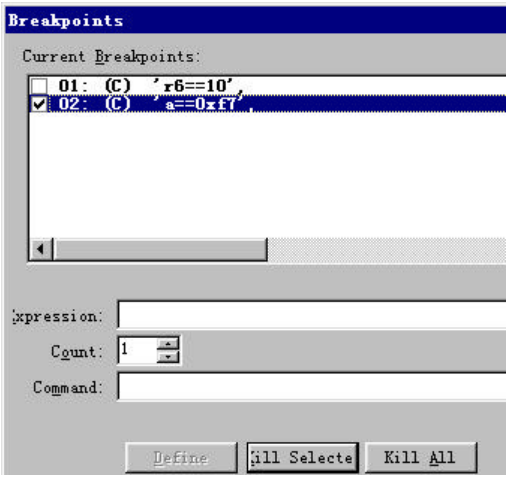

图 4 断点设置对话框

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。

四、实例调试

为进行程序的调试，我们首先给源程序制造一个错误，将延时子程序的第三行"DJNZ R6,$"后的$改为 D1，然后重新编译，由于程序中并无语法错误，所以编译时不会有任何出错提示，但由于转移目的地出错，所以子程序将陷入无限循环中。

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执行完后将停止，这个结果与预期不同，可以看出所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"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"DJNZ R6,D1"，打开在线汇编的对话框，将程序改为"DJNZ R7,0EH"，即转回本条指令所在行继续执行，其中 0EH 是本条指令在程序存储器中的位置，这个值可以通过在线汇编窗口看到，如图 3 所示。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。