



实验 2 任务 1 调试练习举例

1. 查看一下项目的编译优化级是 off。
2. 先点击  运行一次程序，但看不到预期的现象，点击  暂停程序，发现程序停在不是 main()，也不是 delay() 函数处，停在了一个 abort() 函数内，说明程序跑飞，不受自己编写的程序控制了。一看程序，从结构上，缺少了主循环，所以赶紧如文本框 1 加上 while(1){ } 主循环。

//文本框1

```
while(1){           //主循环
    P1OUT=~P1OUT;    //将端口1的值取反后输出
    delay( );        //调用函数延时
}
```

3. 重新编译连接，出现图 2D-1 的提示程序修改，是否重新加载的界面，点击 yes，重新下载编译连接好的程序到单片机中。如果不出现图 2D-1 界面，可点击 Run>Debug，重新下载程序到单片机中。

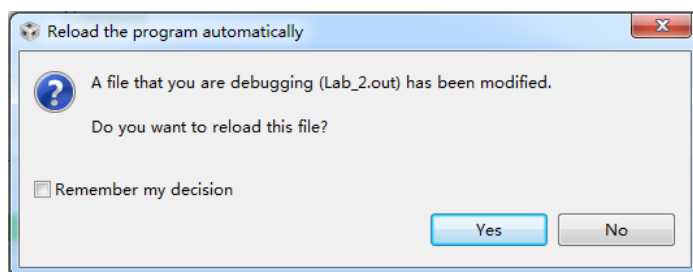







图 2D-1 提示程序修改后提示是否重新加载的界面

4. 在 Debug 窗口，点击  运行程序，仍看不到预想的现象。点击  暂停程序，指针指在程序的某条语句上，本例一般停在 delay() 函数处，说明程序不跑飞了。点击  复位程序，准备重新执行。因为程序比较短，可点击  或键盘上按下 F6，采用单步（step over）方式一条一条语句运行程序，当运行到 P1OUT=~P1OUT 语句时，连接的 LED 灯均没有反应。
仔细检查接线和程序，发现接线是 P2.0~P2.7，但程序控制的是端口 1，程序与接线不对应，把程序中的 P1 改 P2，如文本框 2。

//文本框 2

```
P2DIR=0xff;         //设置端口 2 为输出
while(1){
    P2OUT=~P2OUT;    //将端口 2 的值取反后输出
    delay( );        //调用函数延时
}
```

5. 为加快查找问题，不用单步执行整个程序，改用断点方式控制程序执行。如图 2D-2 双击语句 P2OUT=~P2OUT 最左侧，在此处加一个断点，然后点击  运行程序，可以看到灯有亮灭，开始有现象，但不是同时亮同时灭。应该没有设置 P2OUT 初值造成。如文本框 3，在 while(1) 主循环前加上 P2OUT=0。

```

main.c
1 #include "msp430.h"
2 void delay( );
3 int main ( void )
4 {
5     WDCTL = WDTPW + WDTHOLD; //关闭看门狗
6     P2DIR=0xff; //设置端口2为输出
7     while(1){
8         P2OUT=~P2OUT; //将端口2的值取反后输出
9         delay( ); //调用函数延时
10    }

```

图 2D-2 设置断点调试

//文本框 3

```


P2OUT=0; //设置显示初值
P2DIR=0xff; //设置端口 2 为输出
while(1){
    P2OUT=~P2OUT; //将端口 2 的值取反后输出
    delay( ); //调用函数延时
}

```

- 重新编译连接下载后，进入 DEBUG，此时断点还在语句 `P2OUT=~P2OUT` 处，运行程序，发现 LED 灯 L8 的显示亮度异常，且不该亮时也半亮。因其连接的引脚是 P2.7，如图 2D-3，用 View/Registers 查看与端口 P2 有关的端口寄存器，发现 P2SEL 的 P6、P7 两位上电复位值不为 0，而是 1，从而使 P2.6、P2.7 上电复位后其功能不为基本输入输出，而是其他功能，故不能正确控制连接的 LED 灯 L7、L8。

Registers	
Name	Value
P2OUT	0x00
P2DIR	0xFF
P2IFG	0x3F
P2IES	0xFF
P2IE	0x00
P2SEL	0xC0
P7	1
P6	1
P5	0
P4	0
P3	0
P2	0
P1	0
P0	0
P2SEL2	0x00

图 2D-3 查看引脚相关 IO 寄存器的值

7. 在程序入口处加上对 P2SEL、P2SEL2 的初始化设置，如文本框 4，并保持图 2D-2 中 P2OUT=~P2OUT 前的断点设置，反复点击  运行程序，观察到现象对了：8 个灯同亮，同时灭。


//文本框 4


P2SEL=0; //设置端口P2的8个引脚为基本输入输出功能

P2SEL2=0;

P2OUT=0; //设置显示初值

P2DIR=0xff; //设置端口 2 为输出

8. 双击语句 P2OUT=~P2OUT 的最左端，取消 P2OUT=~P2OUT 前的断点，再点击  运行程序，发现 8 个灯一直全亮，看不到全亮、全灭交替循环的现象。说明，P2OUT=~P2OUT 后 delay() 函数延时时间太短。将 **for** (j=0;j<0x5;j++); 中的 5 改为 0xff，再运行，现象连续运行、断电运行的现象和之前相同，延时时间还是太短。

9. 将 **for** (j=0;j<0xff;j++)改为 j<0xffff，运行程序，仍然看不到现象，且即使在 P2OUT=~P2OUT 前加断点，也看不到现象。同时发现程序不会在设置的断点 P2OUT=~P2OUT 处停下，也就是程序一直处于运行状态。当点击  暂停运行，程序总是停在 delay() 函数处，说明程序没有跑飞。双击 delay() 函数的语句 **for** (j=0;j<0xffff;j++)的最左端，在此添加一个断点。并且如图 2D-4，用 View/Express 或 View/Variables，查看变量 j。





(x)= Variables 1010 0101 Registers			
Name	Type	Value	Location
(x)= j	unsigned char	231 '\xe7'	0x03FA

图 2D-4 查看变量的值

10. 可点击图 2D-4 中 j 的 Value 处，将变量 j 的值改为 0xff，然后用 F6 单步执行程序，发现此时 j 的值从 0xff 变成了 0，也就是 j 的最大是 0xFF，永远<0xFFFF，for 循环的条件永远成立，这样，delay（）函数永远在执行，不会结束。从而也就不会再循环执行语句 P2OUT=~P2OUT，产生 8 个 LED 全亮、全灭的变化。

原因是 j 为 unsigned char 类型，即无符号 char 类型，其最大值为 0xff，而 0xff<0xffff 永远成立。

11. 将 j 的类型改为 unsigned int，再运行程序。语句 P2OUT=~P2OUT 前有断点和没断点，现象都正确。至此程序基本调试完毕。
12. 如果将上面调试好的程序，将编译优化级从 off 改为 1-Registers 级，会发现现象又不对了。所以，建议做实验时，将编译优化级置为 off，即编译时对程序不做太多优化处理。

通过上面调试练习，掌握  resume(运行)、 suspend(暂停)、 restart（复位）、 F6 (step over) 和断点设置等几种执行命令，并能灵活应用它们控制程序的运行，结合查看寄存器、以及变量的值，定位和查找程序中的问题，调试出正确的程序。