第 1 章 ARM CORTEX-M3 程序进入错误中断调试笔记

用 ARM CORTEX-M3 内核的 MCU 做开发，在调试程序时经常遇到会遇到程序进入错误中断的情况，图 1- 1 所示。

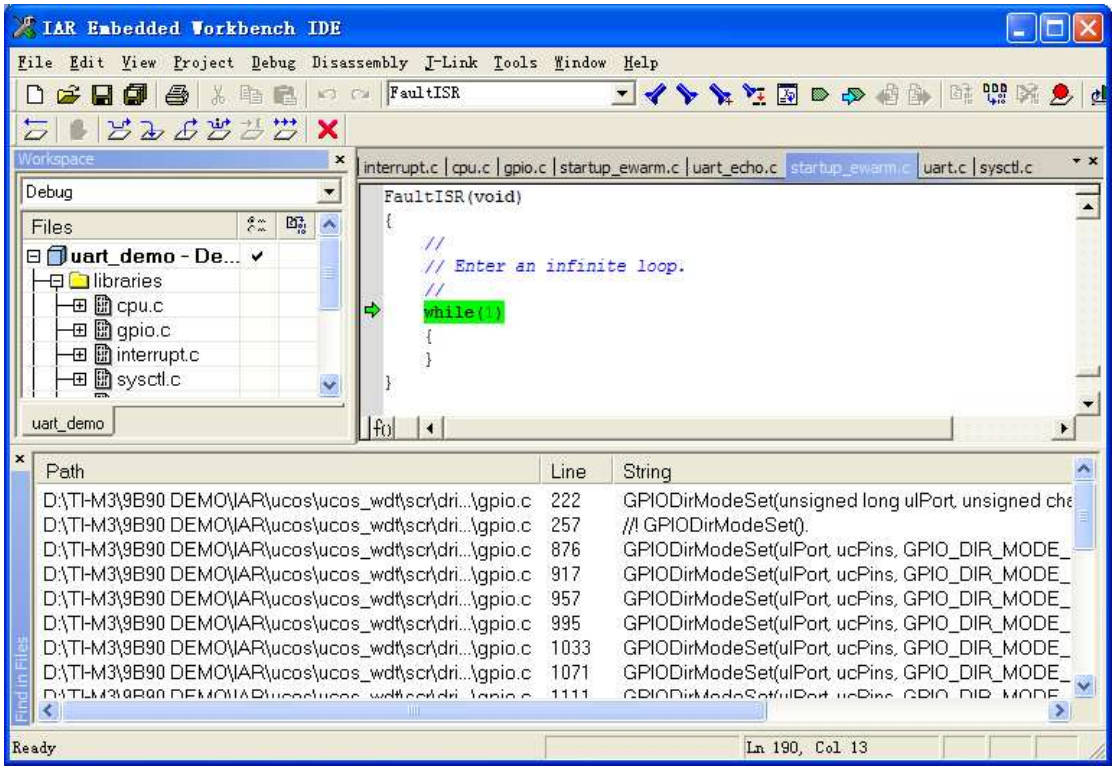

图 1- 1 程序进入错误中断

遇到这种情况，很多朋友茫然不知所措，因为目前主流的开发环境 IAR 和 KEIL 都只提供单步跟踪的功能，而没有轨迹跟踪或单步后退的功能。如果程序较小或程序进入 main() 函数不久就进入错误中断，还可以通过单步跟踪的方法找出问题。但如果程序很大或程序运行很久后才进入错误中断就非常头痛了，不知程序是从哪里跳进了错误中断，更不知道引发错误中断的原因。

本人在使用 TI 的 ARM CORTEX-M3 过程中也曾遇到这些问题，并总结出了一些经验，在此无私奉献给大家，为大家解决问题提供一个思路。

声明：此方法对内存泄露不管用。

言归正传，我们知道 ARM CORTEX-M3 进入中断时，内核自动将 R0,R1,R2,R3,R12,LP,PC,XPSR 8 个寄存器压入堆栈，图 1- 2 所示。而在退出中断程序时，内核自动从堆栈中恢复这 8 个寄存器的值。

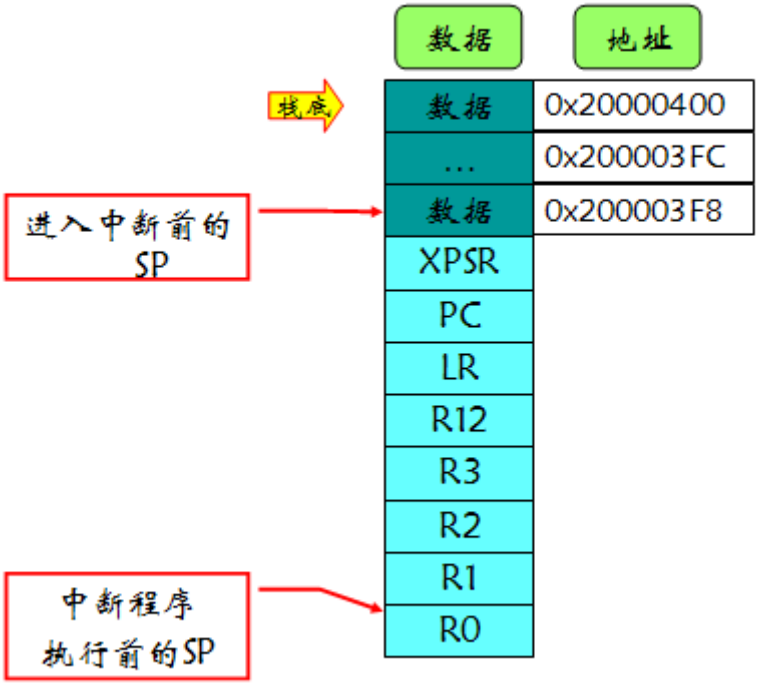

图 1- 2 中断时 ARM CORTEX-M3 内核自动压栈

我们知道，**PC** 总是指向正在取值的指令，那么在进入错误中断时，堆栈中保存的 **PC** 指向进入错误中断前执行的最后一条指令。

LR 用于保存子程序的返回地址，那么在进入错误中断时，堆栈中保存的 **LR** 的值即进入错误中断前执行的函数地址。

由此可以看出，我们可以通过堆栈中的 PC 与 LR 的值找出进入错误中断前执行的最后一条指令与执行的最后一个函数，从而可以定位出发生错误的地点，进而可以分析出引发错误的原因。

当前主流 ARM 编译，如 KEIL，IAR 等都是使用**满递减**的堆栈增长方式，即堆栈是从高地址开始，向低地址增长，图 1- 2 所示。

我们知道在错误中断中，程序一直在做死循环，所以进入中断程序后，SP 的值是没有变化的。

由此我们可以得出堆栈中保存的 **LR** 的计算公式

LR = **SP** + 5*4

5 是因为 **LR** 是进入中断时，内核压栈时，从 SP 往上第 6 个数据

4 是因为内核压栈时是 4 字节对齐

LR = **SP** + 20

由此我们可以得出堆栈中保存的 **PC** 的计算公式

PC = **SP** + 6*4

6 是因为 **PC** 是进入中断时，内核压栈时，从 SP 往上第 7 个数据

4 是因为内核压栈时是 4 字节对齐

PC = **SP** + 24

需要注意的是 ARM CORTEX-M3 内核的堆栈分为**主堆栈(MSP)**和**进程堆栈(PSP)**。在没有使用 OS 的情况下，中断程序与普通程序都使用 **MSP**；在使用 OS 的情况下，中断程序使用 **MSP**，而进程程序使用 **PSP**。

我们以一个实例来说明，如程序清单 1- 1 所示，这是本来是一个完整的串口初始化函数，但我特意屏蔽掉了 L（1）语句，由于没有使能 UART0 外设，这样程序在执行 L（2）时会进入错误中断。

程序清单 1- 1 示例程序

```
int
main(void)
{
    // Set the clocking to run directly from the crystal.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_16MHZ);
    //
    // Enable processor interrupts.
    IntMasterEnable();

    //  使能 UATR, 配置 UART IO
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    // SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);                    L（1）

    //
    // Set GPIO A0 and A1 as UART pins.
    //
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);          L（2）

    //
    // Configure the UART for 115,200, 8-N-1 operation.
    //
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));
    //
    // Enable the UART interrupt.
```

```
    //
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    //
    // Prompt for text to be entered.
    UARTSend((unsigned char *)"Enter text: ", 12);

    // Loop forever echoing data through the UART.
    while(1)
    {
    }
}
```

在进入错误中断后，我们可以寄存器串口看到 **MSP** 的值是 0x200000c8，如图 1- 3 所示。

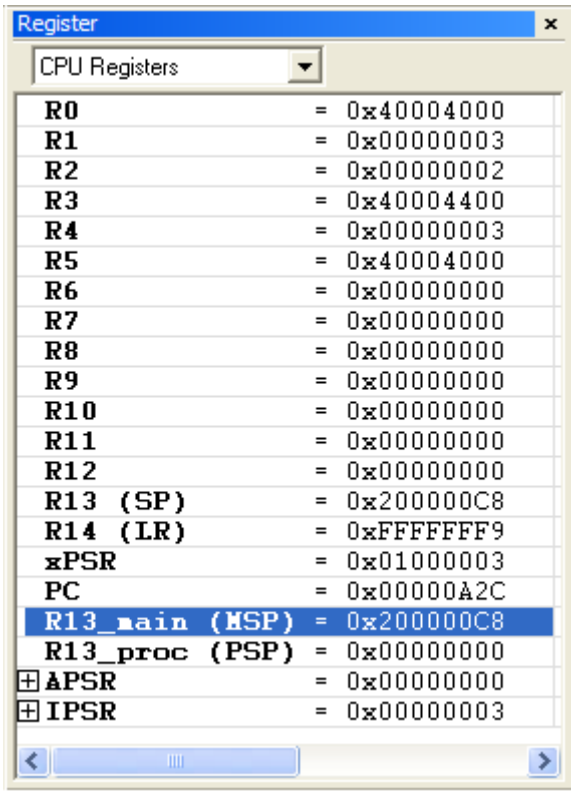

图 1- 3 寄存器窗口

由于本例程没有使用 OS，所以 MSP 就是 SP 的值。根据上面的公式

LR = **SP** + 20

= 0x200000c8 + 20

= 0x200000dc

我们在 MEMERY 窗口，输入 0x200000dc，可以看出 0x200000dc 地址的值是 0x00000707，如图 1- 4 所示。这个 0x00000707 就是最后执行的函数的地址。

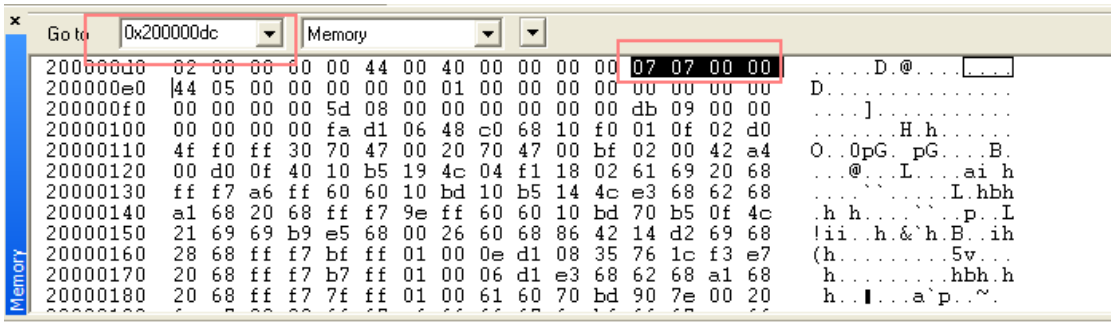

图 1- 4 MEMORY 窗口

得到最后执行的函数的地址后，我们在汇编程序窗口，输入 0x00000707，IAR 开发环境就自动跳转到对应的汇编函数，如图 1- 5 所示。

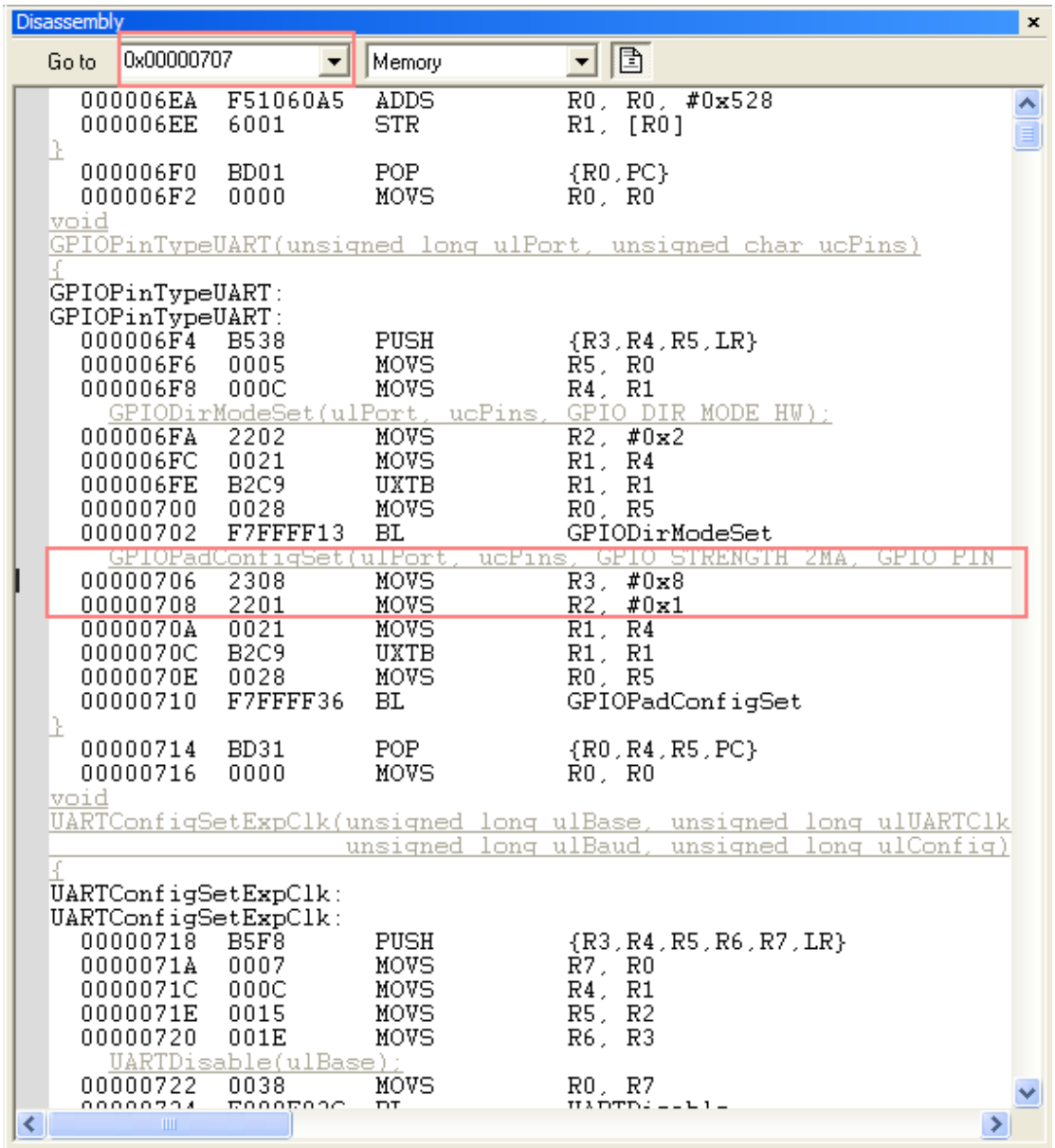

图 1- 5 汇编窗口

根据汇编窗口提示，进入错误中断时最后调用的一个函数是 void GPIOPinTypeUART(unsigned long ulPort, unsigned char ucPins)，即 L（2）函数调用的第一个函数。