restricted access to resources. Upon returning from the exception, the processor will revert back to the state from which it left, so there is no way for a user program to change the privilege level by simply changing a bit. It must use an exception handler (which forces the processor into a privileged level) that controls the value in the CONTROL register.

EXAMPLE 15.1

We'll return to this example later in the chapter, with some modifications along the way, as it demonstrates the various aspects of exception handling in the Cortex-M4. Let's begin by building a quick-and-dirty routine that forces the processor into privileged Handler mode from privileged Thread mode. In Chapter 7, the idea of trapping division by zero was only mentioned, leaving an actual case study until now. If you type the following example into the Keil tools, using a Tiva TM4C1233H6PM as the target processor, and then single-step through the code, just out of reset the processor will begin executing the instructions after the label Reset_Handler. Note that many of the registers are memory mapped. For the full list of registers, see the Tiva TM4C1233H6PM Microcontroller Data Sheet (Texas Instruments 2013b).

```
Stack        EQU     0x00000100
DivbyZ       EQU     0xD14
SYSHNDCTRL   EQU     0xD24
Usagefault   EQU     0xD2A
NVICBase     EQU     0xE000E000

     AREA    STACK, NOINIT, READWRITE, ALIGN=3
StackMem
     SPACE   Stack
     PRESERVE8

     AREA RESET, CODE, READONLY
     THUMB

; The vector table sits here
; We'll define just a few of them and leave the rest at 0 for now

     DCD     StackMem+Stack        ; Top of Stack
     DCD     Reset_Handler         ; Reset Handler
     DCD     NmiISR                ; NMI Handler
     DCD     FaultISR              ; Hard Fault Handler
     DCD     IntDefaultHandler     ; MPU Fault Handler
     DCD     IntDefaultHandler     ; Bus Fault Handler
     DCD     IntDefaultHandler     ; Usage Fault Handler

     EXPORT Reset_Handler
     ENTRY

Reset_Handler
           ; enable the divide-by-zero trap
           ; located in the NVIC
           ; base: 0xE000E000
           ; offset: 0xD14
```

```
            ; bit: 4
            LDR         r6, =NVICBase
            LDR         r7, =DivbyZ
            LDR         r1, [r6, r7]
            ORR         r1, #0x10           ; enable bit 4
            STR         r1, [r6, r7]

            ; now turn on the usage fault exception
            LDR         r7, =SYSHNDCTRL (p. 163)
            LDR         r1, [r6, r7]
            ORR         r1, #0x40000
            STR         r1, [r6, r7]

            ; try out a divide by 2 then a divide by 0!
            MOV         r0, #0
            MOV         r1, #0x11111111
            MOV         r2, #0x22222222
            MOV         r3, #0x33333333

            ; this divide works just fine
            UDIV        r4, r2, r1
            ; this divide takes an exception
            UDIV        r5, r3, r0

Exit        B       Exit

NmiISR      B       NmiISR
FaultISR    B       FaultISR
IntDefaultHandler

            ; let's read the Usage Fault Status Register

            LDR         r7, =Usagefault
            LDRH        r1, [r6, r7]
            TEQ         r1, #0x200
            IT          NE
            LDRNE       r9, =0xDEADDEAD
            ; r1 should have bit 9 set indicating
            ; a divide-by-zero has taken place
done        B           done
            ALIGN

            END
```

Continue single-stepping through the MOV and LDR instructions until you come to the first of the two UDIV (unsigned divide) operations. If you examine the registers and the state information using the Keil tools, you see that the first divide instruction is perfectly legal, and it will produce a value in register r2. More importantly, the machine is operating in Thread mode and it is privileged, shown in Figure 15.3. If you try to execute the next divide instruction, one which tries to divide a number by zero, you should see the machine change modes to Handler mode. The program has enabled a particular type of exception (usage faults, which we'll cover in Section 15.6) and enabled divide-by-zero traps so that we can watch the