Interactive tool 12.2 shows the context switch from executing in the foreground to running a periodic SysTick ISR. Before the interrupt occurs, the I bit in the PRIMASK is 0 signifying interrupts are enabled, and the interrupt number (ISRNUM) in the **IPSR** register is 0, meaning we are running in **Thread mode** (i.e., the main program, and not an ISR). **Handler mode** is signified by a nonzero value in **IPSR**. When **BASEPRI** register is zero, all interrupts are allowed and the **BASEPRI** register is not active.

When the SysTick counter goes from 1 to 0, the **Count** flag in the **NVIC_ST_CTRL_R** register is set, triggering an interrupt. The current instruction is finished. (a) Eight registers are pushed on the stack with **R0** on top. These registers are pushed onto the stack . (b) The vector address is loaded into the **PC** ("Vector address" column in Table 12.1). (c) The **IPSR** register is set to 15 ("Number" column in Table 12.1) (d) The top 24 bits of **LR** are set to 0xFFFFFF, signifying the processor is executing an ISR. The bottom eight bits specify how to return from interrupt.

0xE1 Return to Handler mode MSP (using floating point state on TM4C)
0xE9 Return to Thread mode MSP (using floating point state on TM4C)
0xED Return to Thread mode PSP (using floating point state on TM4C)
0xF1 Return to Handler mode MSP
0xF9 Return to Thread mode MSP ← *in this class we will always be using this one*
0xFD Return to Thread mode PSP

After pushing the registers, the processor always uses the main stack pointer (**MSP**) during the execution of the ISR. Events 2, 3, and 4 can occur simultaneously

---

Start Click Start to start the context switch by pushing the current registers into the stack.

To **return from an interrupt**, the ISR executes the typical function return **BX LR**. However, since the top 24 bits of **LR** are 0xFFFFFF, it knows to return from interrupt by popping the eight registers off the stack. Since the bottom eight bits of **LR** in this case are 0b11111001, it returns to thread mode using the **MSP** as its stack pointer. Since the **ISPR** is part of the **PSR** that is popped, it is automatically reset its previous state.

A **nested interrupt** occurs when a higher priority interrupt suspends an ISR. The lower priority interrupt will finish after the higher priority ISR completes. When one interrupt preempts another, the **LR** is set to 0xFFFFFFF1, so it knows to return to handler mode. **Tail chaining** occurs when one ISR executes immediately after another. Optimization occurs because the eight registers need not be popped only to be pushed once again. If an interrupt is triggered and is in the process of stacking registers when a higher priority interrupt is requested, this **late arrival interrupt** will be executed first.

Priority determines the order of service when two or more requests are made simultaneously. Priority also allows a higher priority request to suspend a lower priority request currently being processed. Usually, if two requests have the same priority, we do not allow them to interrupt each other. NVIC assigns a priority level to each interrupt trigger. This mechanism allows a higher priority trigger to interrupt the ISR of a lower priority request. Conversely, if a lower priority request occurs while running an ISR of a higher priority trigger, it will be postponed until the higher priority service is complete.

Observation: There are many interrupt sources, but an effective system will use only a few.

Program 12.3 gives the definitions in **startup.s** that allow the software to enable and disable interrupts. These functions are callable from either assembly or C code. The wait for interrupt can be used to place the processor in low-power sleep mode while it waits for an interrupt.

```
;*********** DisableInterrupts ***************
; disable interrupts
; inputs:  none
; outputs: none
DisableInterrupts  CPSID  I    ;set I=1
          BX    LR

;*********** EnableInterrupts ***************
; enable interrupts
; inputs:  none
; outputs: none
EnableInterrupts   CPSIE  I    ;set I=0
          BX    LR

;*********** WaitForInterrupt ************************
; go to low power mode while waiting for the next interrupt
; inputs:  none
; outputs: none
WaitForInterrupt
        WFI
        BX      LR
```

Program 12.3. Assembly functions needed for interrupt enabling and disabling.

About (https://www.edx.org/about-us) Jobs (https://www.edx.org/jobs)

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.

(http://www.meetup.com/edX-Global-Community/)

(http://www.facebook.com/EdxOnline)

(https://twitter.com/edXOnline)

(https://plus.google.com/108235383044095082735/posts)

(http://youtube.com/user/edxonline)

© 2014 edX, some rights reserved.

Terms of Service and Honor Code -
Privacy Policy (https://www.edx.org/edx-privacy-policy)

Help