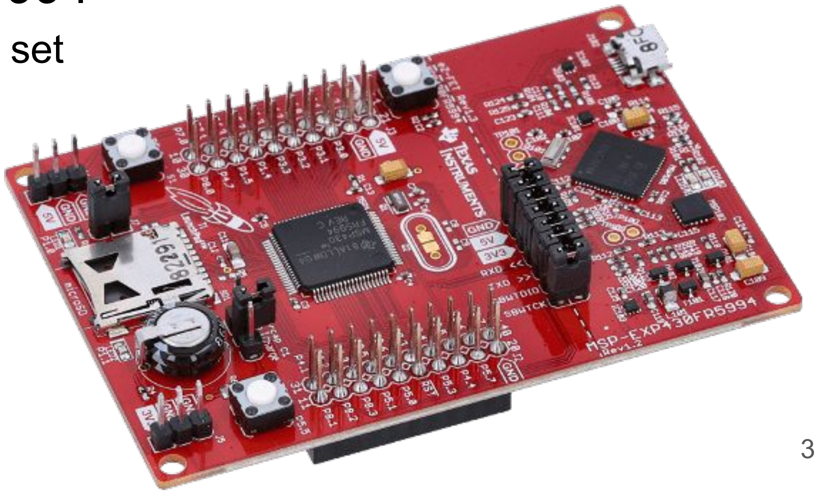
FreeRTOS on MSP430

Prof. Li-Pin Chang
ESSLAB@NYCU

Introduction to TI MSP430 uP

MSP430FR5994 LaunchPad

- MSP430 CPU
  - 16 bit ultra-low power micro-processor
  - Designed by Texas Instrument
  - Clock rate 8MHz by default (max 24MHz)
  - Can address up to 1MB of working memory in the extended mode (20 bits addressing)
- MSP-EXP430FR5994 LaunchPad
  - LaunchPad = development kit
  - Based on MSP430FR5994 SoC (MSP430 inside)
  - Working memory: 16KB of SRAM + 256KB of FRAM
- HowTo: run FreeRTOS on MSP430FR5994
  - Check “FreeRTOS Environment Setup” slice set

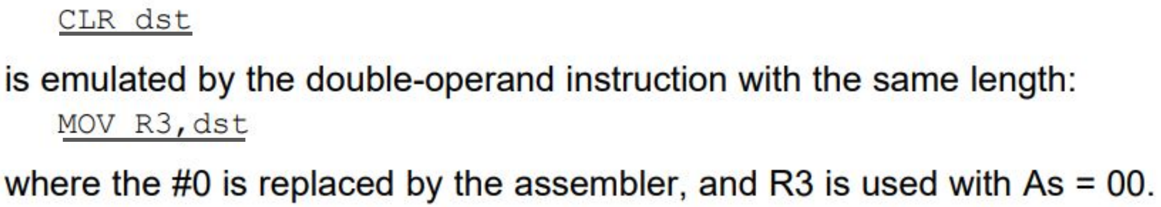


MSP430 Register Architecture

- General purpose registers (R4 ~ R15)
    - contain 8-bit, 16-bit, 20-bit values
- Special purpose registers (R0~R3)
    - R0: Program Counter (PC)
        - Points to the next instruction to be executed
    - R1: Stack Pointer (SP)
        - Pointer to the stack
    - R2: Status Register (SR)
        - Arithmetic operation flags such as carry, zero, etc.
        - GIE: this bit is set, it enables maskable interrupt
    - R3: Constant Generator (CG)
        - Pre-configured to be one among the pre-defined bit patterns, e.g., 0xffff or 0x0000
        - Avoid memory reference and reduce instruction size

MSP430 Register Architecture

- General purpose registers (R4 ~ R15)
  - contain 8-bit, 16-bit, 20-bit values
- Special purpose registers (R0~R3)
  - R0: Program Counter (PC)
    - Points to the next instruction to be executed
  - R1: Stack Pointer (SP)
    - Pointer to the stack
  - R2: Status Register (SR)
    - Status of the result of the operation
      - {C, Z, N}: this bit is set when the result {has carry, is 0, is negative}
      - GIE: this bit is set, it enables maskable interrupt
      - V: this bit is set when the result is overflowed
  - R3: Constant Generator (CG)
    - Allows assembler to support additional instructions without additional code word
      - example:
        - additional 16-bit word “0” is not needed in program code



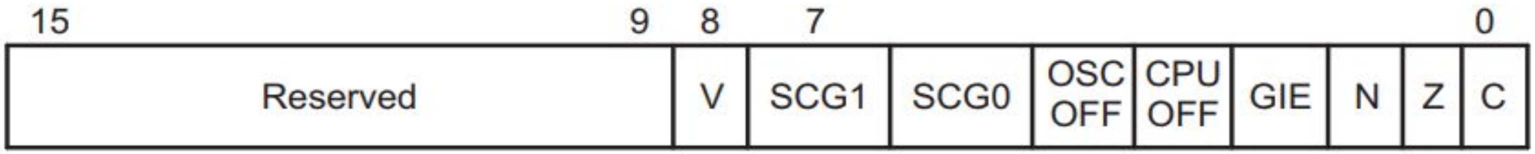
```
CLR dst
```

is emulated by the double-operand instruction with the same length:

```
MOV R3,dst
```

where the #0 is replaced by the assembler, and R3 is used with As = 00.

Status Register (R2)



15 9	8	7							0
Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

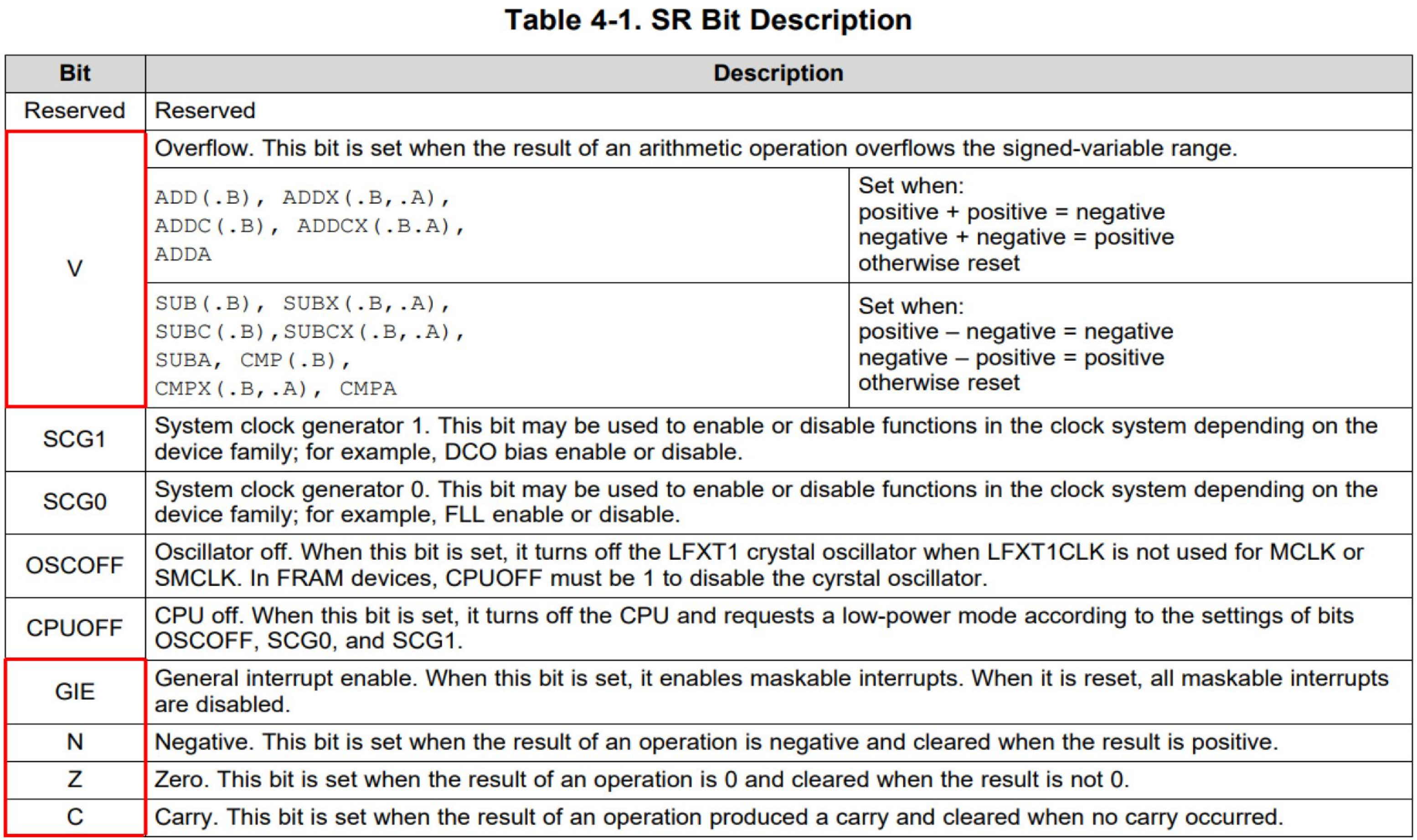


Table 4-1. SR Bit Description

Bit	Description	
Reserved	Reserved	
V	Overflow. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	

	`ADD(.B), ADDX(.B,.A),`
	`ADDC(.B), ADDCX(.B.A),`
	`ADDA`
		Set when:
		positive + positive = negative
		negative + negative = positive
		otherwise reset

	`SUB(.B), SUBX(.B,.A),`
	`SUBC(.B),SUBCX(.B,.A),`
	`SUBA, CMP(.B),`
	`CMPX(.B,.A), CMPA`
		Set when:
		positive – negative = negative
		negative – positive = positive
		otherwise reset

SCG1	System clock generator 1. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, DCO bias enable or disable.	
SCG0	System clock generator 0. This bit may be used to enable or disable functions in the clock system depending on the device family; for example, FLL enable or disable.	
OSCOFF	Oscillator off. When this bit is set, it turns off the LFXT1 crystal oscillator when LFXT1CLK is not used for MCLK or SMCLK. In FRAM devices, CPUOFF must be 1 to disable the cyrstal oscillator.	
CPUOFF	CPU off. When this bit is set, it turns off the CPU and requests a low-power mode according to the settings of bits OSCOFF, SCG0, and SCG1.	
GIE	General interrupt enable. When this bit is set, it enables maskable interrupts. When it is reset, all maskable interrupts are disabled.	
N	Negative. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.	
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

MSP430 Interrupt Basics

- Interrupt vector table
  - Located downward from 0FFFEh (word address)
  - Each vector contains 16-bit address (2 bytes)
- Some example interrupt vectors:
  - Timer_A0 interrupt: 0FFEAh & 0FFE8h
  - GPIO Port 1 interrupt: 0FFDEh

MSP430 Interrupt vector table

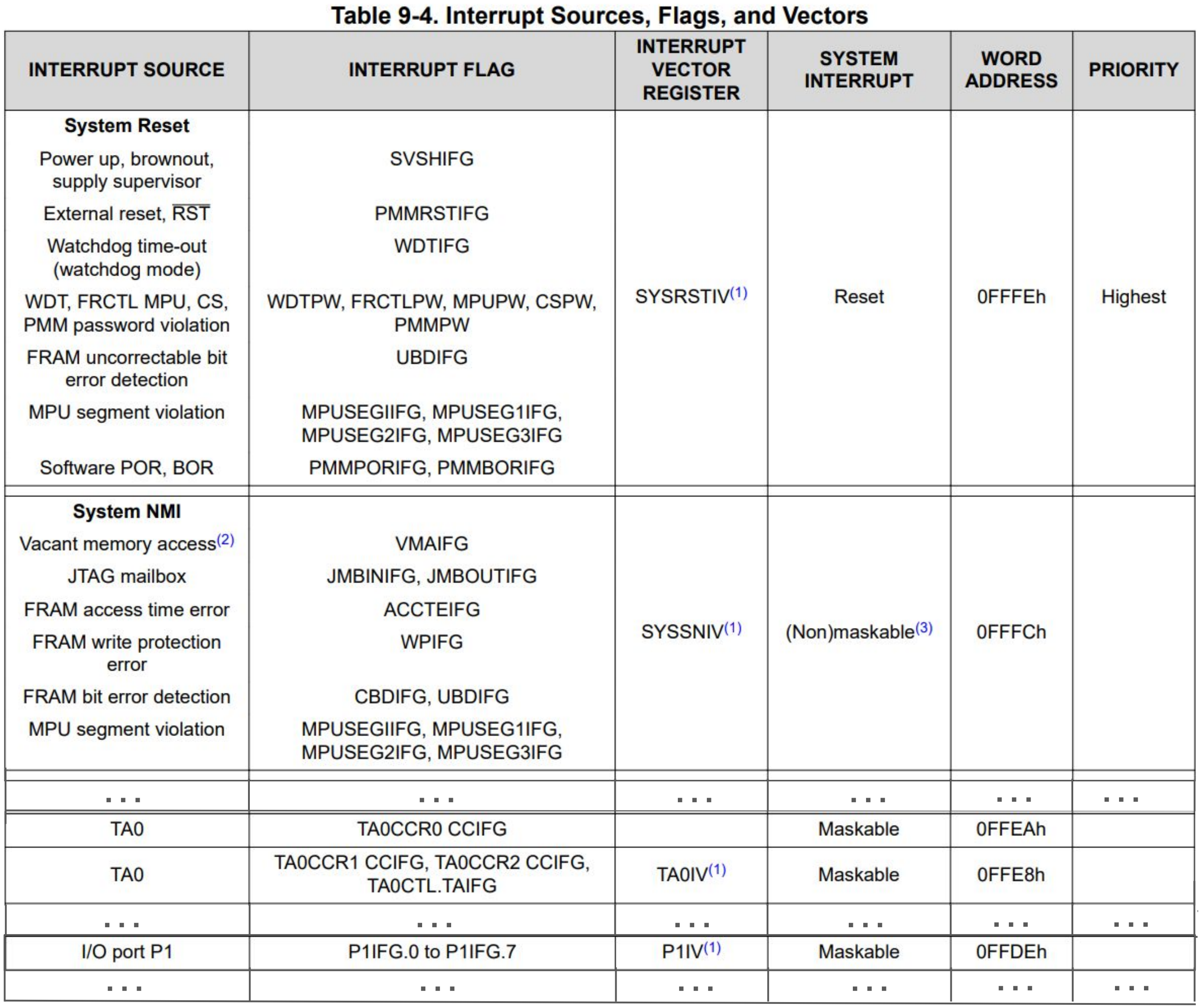


Table 9-4. Interrupt Sources, Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT VECTOR REGISTER	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
System Reset		SYSRSTIV[1]	Reset	0FFFEh	Highest
Power up, brownout, supply supervisor	SVSHIFG				
External reset, $\overline{RST}$	PMMRSTIFG				
Watchdog time-out (watchdog mode)	WDTIFG				
WDT, FRCTL MPU, CS, PMM password violation	WDTPW, FRCTLPW, MPUPW, CSPW, PMMPW				
FRAM uncorrectable bit error detection	UBDIFG				
MPU segment violation	MPUSEGIIFG, MPUSEG1IFG, MPUSEG2IFG, MPUSEG3IFG				
Software POR, BOR	PMMPORIFG, PMMBORIFG				
System NMI		SYSSNIV[1]	(Non)maskable[3]	0FFFCh	
Vacant memory access[2]	VMAIFG				
JTAG mailbox	JMBINIFG, JMBOUTIFG				
FRAM access time error	ACCTEIFG				
FRAM write protection error	WPIFG				
FRAM bit error detection	CBDIFG, UBDIFG				
MPU segment violation	MPUSEGIIFG, MPUSEG1IFG, MPUSEG2IFG, MPUSEG3IFG				
. . .	. . .	. . .	. . .	. . .	. . .
TA0	TA0CCR0 CCIFG		Maskable	0FFEAh	
TA0	TA0CCR1 CCIFG, TA0CCR2 CCIFG, TA0CTL.TAIFG	TA0IV[1]	Maskable	0FFE8h	
. . .	. . .	. . .	. . .	. . .	. . .
I/O port P1	P1IFG.0 to P1IFG.7	P1IV[1]	Maskable	0FFDEh	
. . .	. . .	. . .	. . .	. . .	. . .

Ref.

Interrupt Handling with MSP430

🟦 CPU
🟥 OS

1. Interrupt is acknowledged by the CPU
2. Complete the instruction currently being executed
3. Push PC (R0) and SR (R2) onto the stack
4. Select the highest-priority interrupt among pending ones
5. Acknowledge the interrupt source flag
6. Clear SR to disable interrupts (!)
7. Load the address in the interrupt vector into PC
8. Execute the interrupt service routine (ISR)
9. Return from interrupt (RETI instruction)
    a. pop SR from stack (will re-enable interrupts)
    b. pop PC from stack

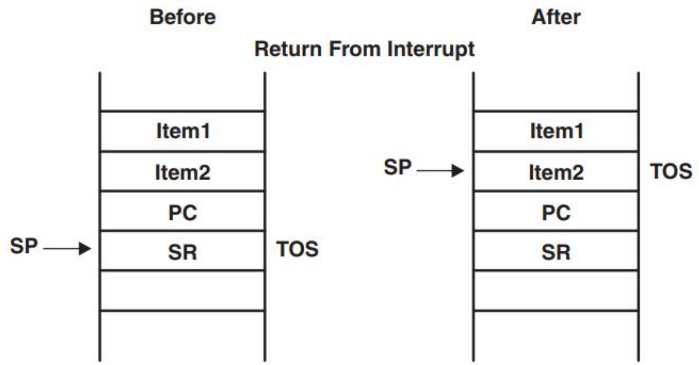
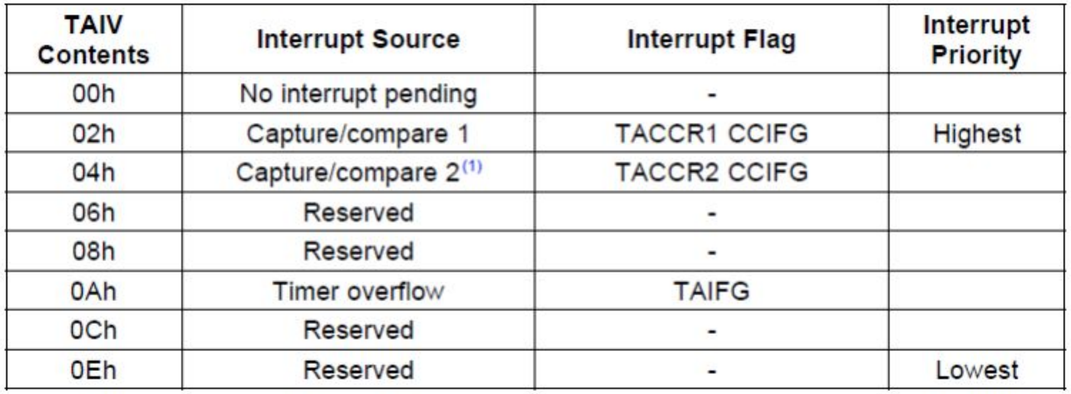


Figure 1-4. Return From Interrupt

Sub Interrupt Events

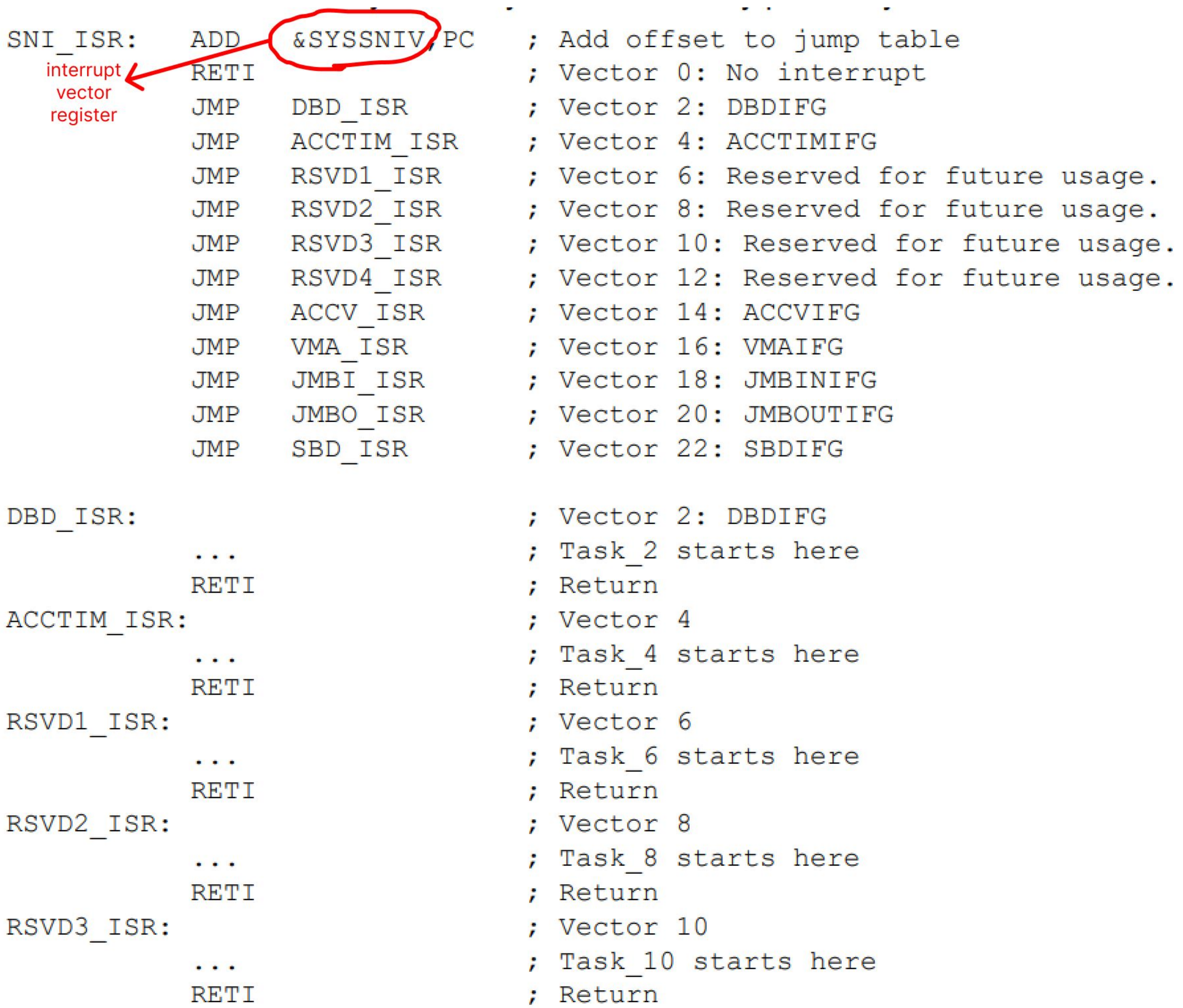
- Related events may share the same interrupt number
  - e.g. Timer interrupt may be generated by
    - Timer count overflow
    - Timer count reaches certain value (TACCR)
    - …
- Identify what causes the interrupt
  - Poll all the related interrupt flags in ISR → slow
  - Read interrupt vector register
- Some interrupt vector registers
  - Timer: TAIV
  - System NMI: SYSSNIV



TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2[1]	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

Sub Interrupt Events: System NMI

Use the value of SYSSNIV as an offset to PC

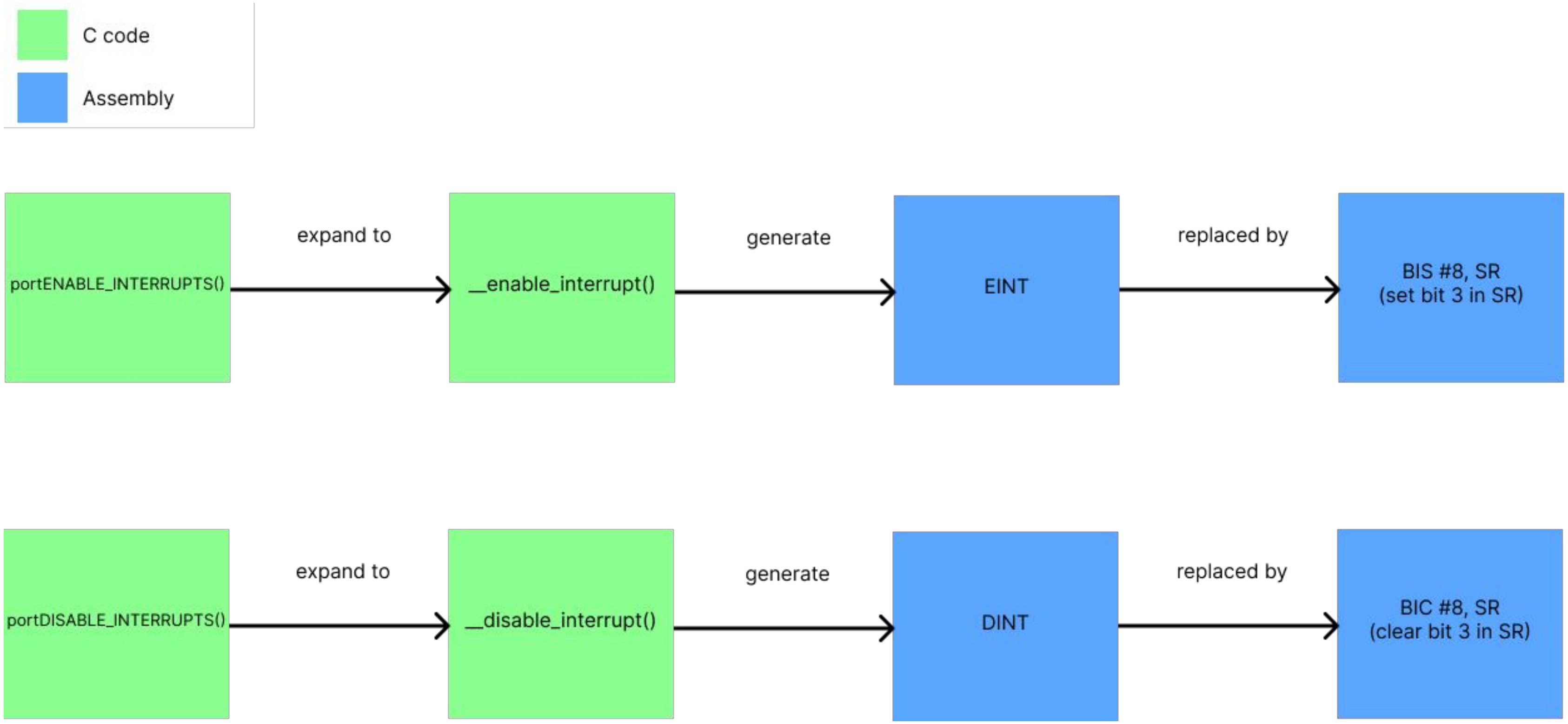


interrupt vector register

```
SNI_ISR:   ADD   &SYSSNIV,PC    ; Add offset to jump table
           RETI                 ; Vector 0: No interrupt
           JMP   DBD_ISR        ; Vector 2: DBDIFG
           JMP   ACCTIM_ISR     ; Vector 4: ACCTIMIFG
           JMP   RSVD1_ISR      ; Vector 6: Reserved for future usage.
           JMP   RSVD2_ISR      ; Vector 8: Reserved for future usage.
           JMP   RSVD3_ISR      ; Vector 10: Reserved for future usage.
           JMP   RSVD4_ISR      ; Vector 12: Reserved for future usage.
           JMP   ACCV_ISR       ; Vector 14: ACCVIFG
           JMP   VMA_ISR        ; Vector 16: VMAIFG
           JMP   JMBI_ISR       ; Vector 18: JMBINIFG
           JMP   JMBO_ISR       ; Vector 20: JMBOUTIFG
           JMP   SBD_ISR        ; Vector 22: SBDIFG

DBD_ISR:                        ; Vector 2: DBDIFG
           ...                  ; Task_2 starts here
           RETI                 ; Return
ACCTIM_ISR:                     ; Vector 4
           ...                  ; Task_4 starts here
           RETI                 ; Return
RSVD1_ISR:                      ; Vector 6
           ...                  ; Task_6 starts here
           RETI                 ; Return
RSVD2_ISR:                      ; Vector 8
           ...                  ; Task_8 starts here
           RETI                 ; Return
RSVD3_ISR:                      ; Vector 10
           ...                  ; Task_10 starts here
           RETI                 ; Return
```

Enable/Disable interrupt on the MSP430

- Enable interrupt:
    - Set global interrupt enable (GIE) bit in SR
- Disable interrupt:
    - Clear global interrupt enable (GIE) bit in SR



🟩 C code

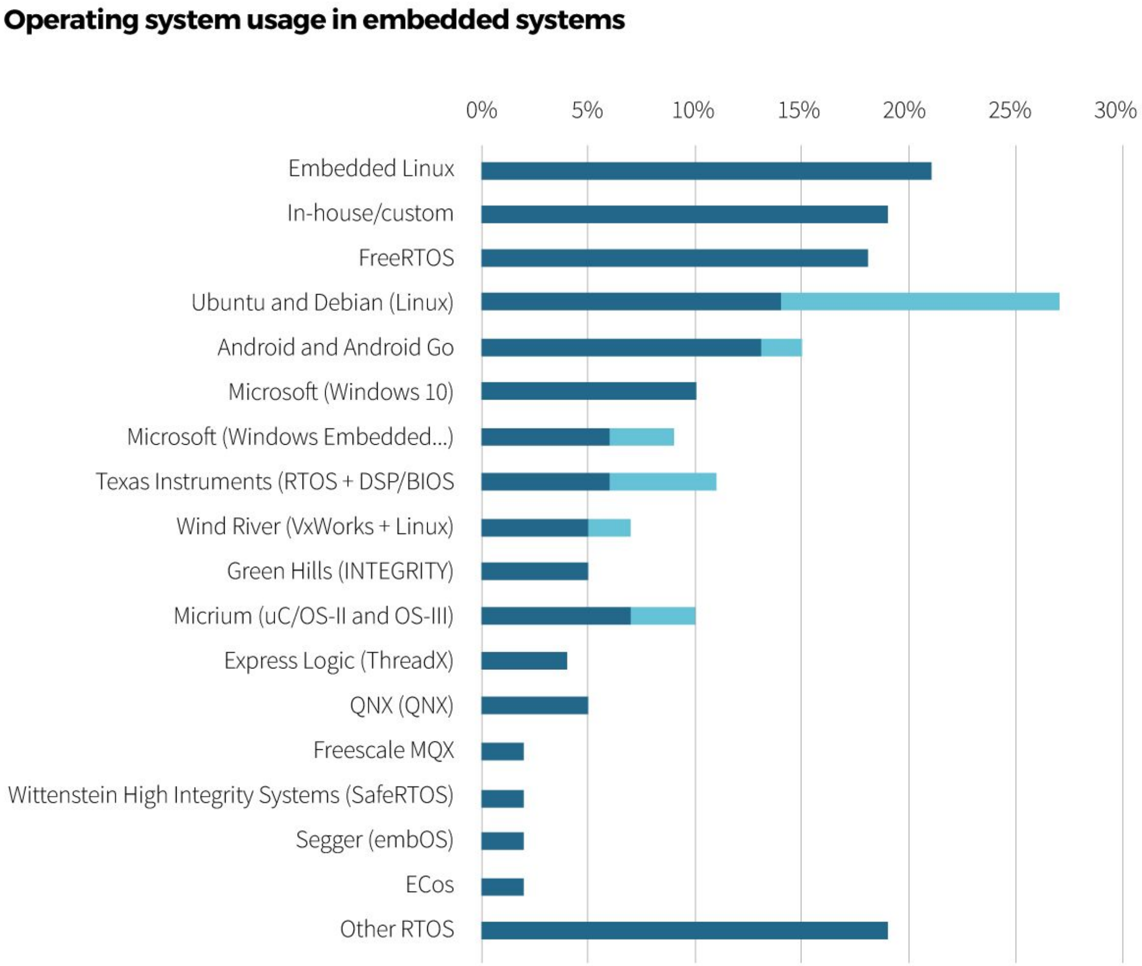
🟦 Assembly

portENABLE_INTERRUPTS() → expand to → __enable_interrupt() → generate → EINT → replaced by → BIS #8, SR (set bit 3 in SR)

portDISABLE_INTERRUPTS() → expand to → __disable_interrupt() → generate → DINT → replaced by → BIC #8, SR (clear bit 3 in SR)

FreeRTOS Kernel

Global RTOS Market Share (2019)



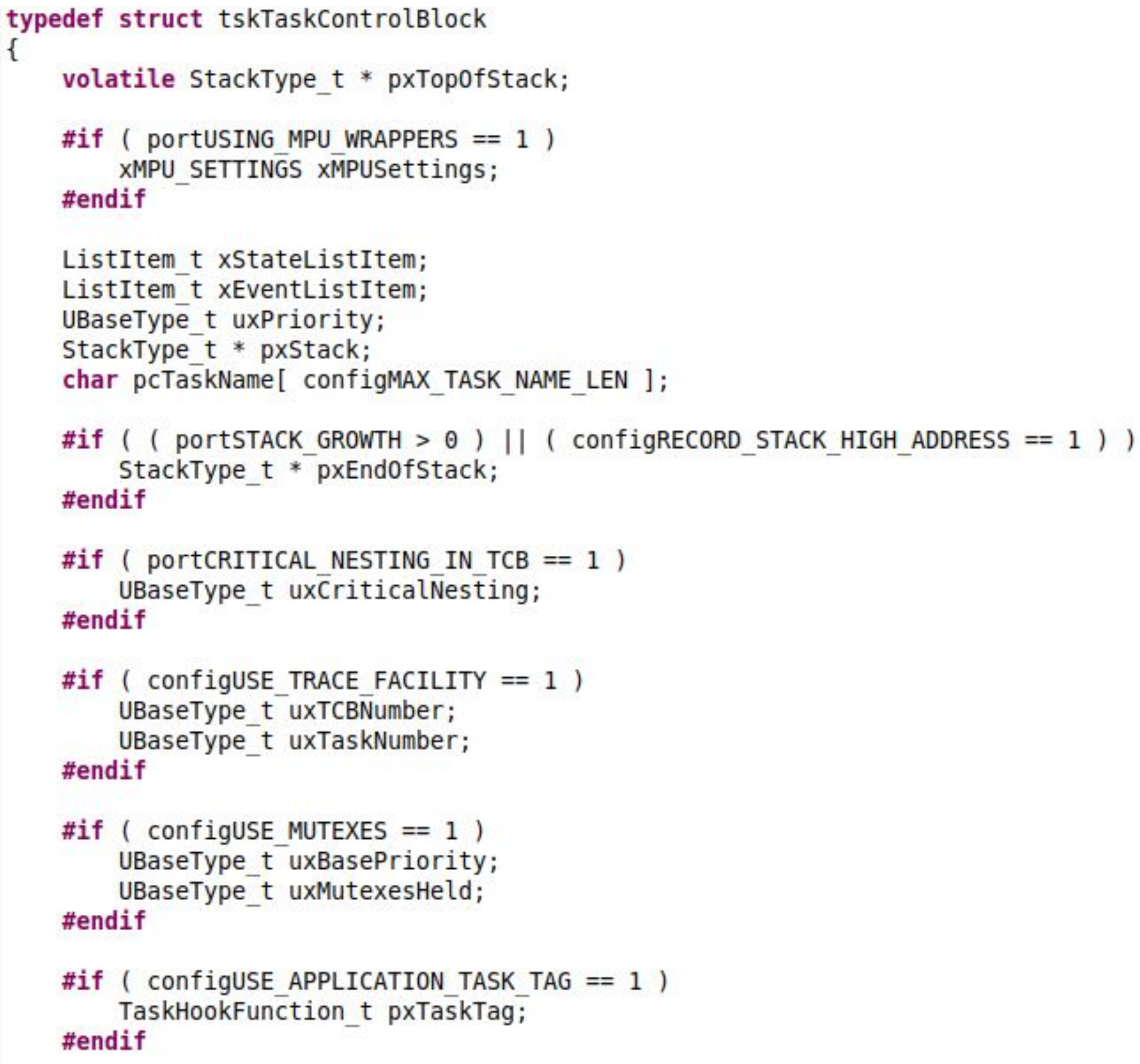
light color = variants

https://www.tuxera.com/blog/data-and-future-of-iiot-os/

Essential Kernel Data Structures

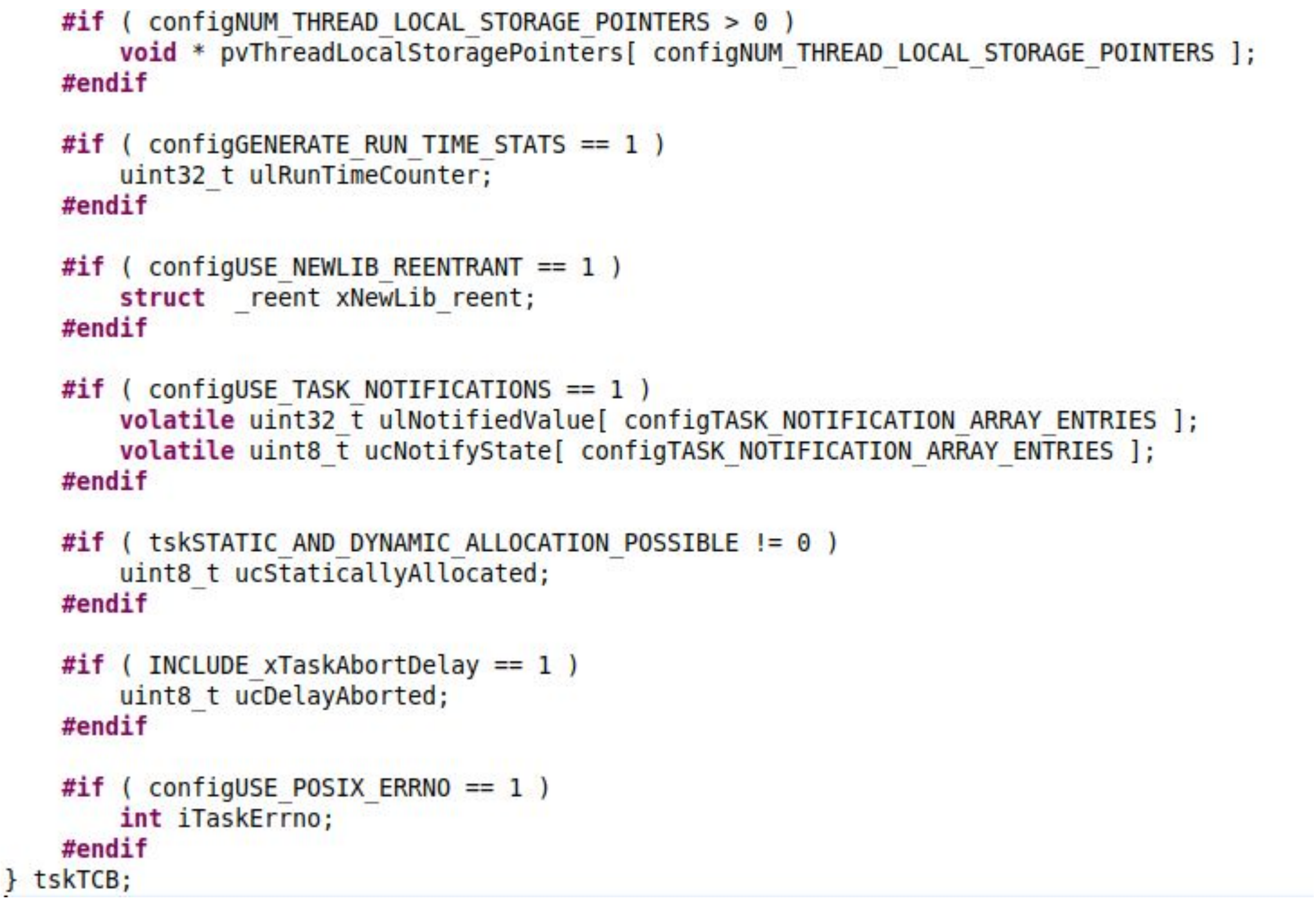
- Task control block
- Task stack
- Ready list
- Delay list

Task Control Block



```c
typedef struct tskTaskControlBlock
{
    volatile StackType_t * pxTopOfStack;

    #if ( portUSING_MPU_WRAPPERS == 1 )
        xMPU_SETTINGS xMPUSettings;
    #endif

    ListItem_t xStateListItem;
    ListItem_t xEventListItem;
    UBaseType_t uxPriority;
    StackType_t * pxStack;
    char pcTaskName[ configMAX_TASK_NAME_LEN ];

    #if ( ( portSTACK_GROWTH > 0 ) || ( configRECORD_STACK_HIGH_ADDRESS == 1 ) )
        StackType_t * pxEndOfStack;
    #endif

    #if ( portCRITICAL_NESTING_IN_TCB == 1 )
        UBaseType_t uxCriticalNesting;
    #endif

    #if ( configUSE_TRACE_FACILITY == 1 )
        UBaseType_t uxTCBNumber;
        UBaseType_t uxTaskNumber;
    #endif

    #if ( configUSE_MUTEXES == 1 )
        UBaseType_t uxBasePriority;
        UBaseType_t uxMutexesHeld;
    #endif

    #if ( configUSE_APPLICATION_TASK_TAG == 1 )
        TaskHookFunction_t pxTaskTag;
    #endif
```

Task Control Block

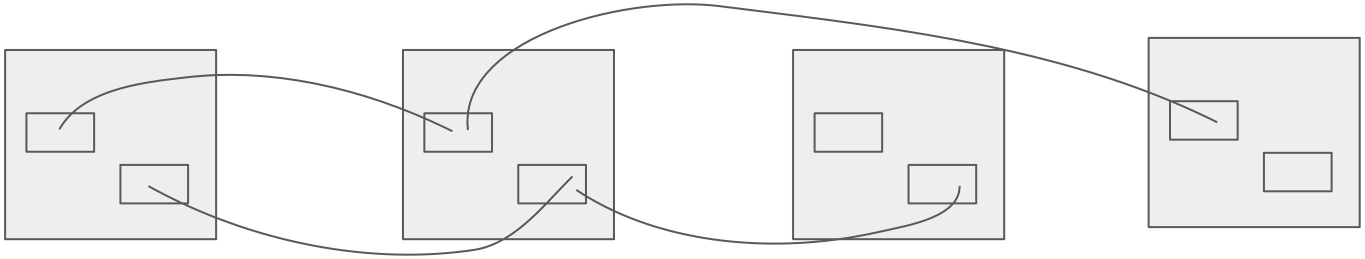


```
    #if ( configNUM_THREAD_LOCAL_STORAGE_POINTERS > 0 )
        void * pvThreadLocalStoragePointers[ configNUM_THREAD_LOCAL_STORAGE_POINTERS ];
    #endif

    #if ( configGENERATE_RUN_TIME_STATS == 1 )
        uint32_t ulRunTimeCounter;
    #endif

    #if ( configUSE_NEWLIB_REENTRANT == 1 )
        struct  _reent xNewLib_reent;
    #endif

    #if ( configUSE_TASK_NOTIFICATIONS == 1 )
        volatile uint32_t ulNotifiedValue[ configTASK_NOTIFICATION_ARRAY_ENTRIES ];
        volatile uint8_t ucNotifyState[ configTASK_NOTIFICATION_ARRAY_ENTRIES ];
    #endif

    #if ( tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE != 0 )
        uint8_t ucStaticallyAllocated;
    #endif

    #if ( INCLUDE_xTaskAbortDelay == 1 )
        uint8_t ucDelayAborted;
    #endif

    #if ( configUSE_POSIX_ERRNO == 1 )
        int iTaskErrno;
    #endif
} tskTCB;
```

Task Control Block

- **pxTopOfStack** points to the location of the last item placed on the tasks stack.
  - This must be the first member of the TCB struct.
- **xStateListItem** : The list that the state list item is reference from denotes the state of that task (Ready, Blocked, Suspended).
- **xEventListItem** : Used to reference a task from an event list.
- **uxPriority** : The priority of the task. 0 is the **lowest** priority.
- **pxStack** points to the start of the stack.
- **pcTaskName** : Descriptive name given to the task when created.
- **pxEndOfStack** points to the highest valid address for the stack.
- **uxBasePriority** : The priority last assigned to the task - used by the priority inheritance mechanism.

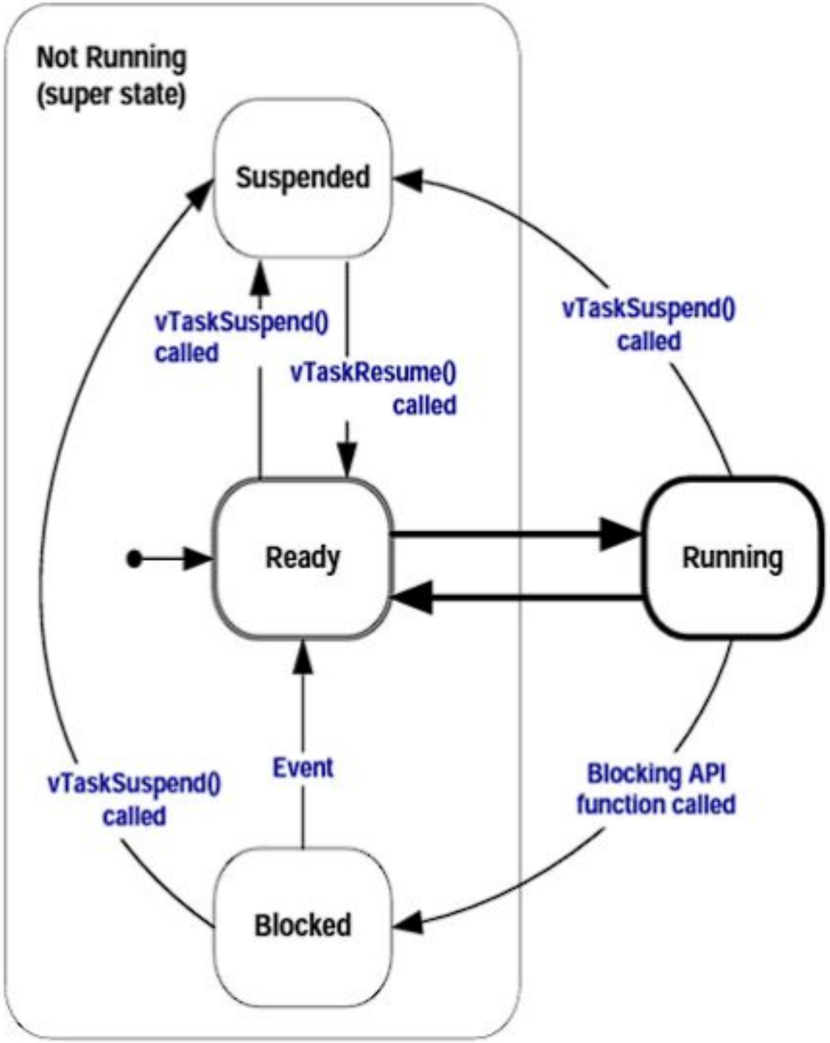
Concept of “list items”



Task States

- Running
  - Only one task can be in the Running state at any one time.
- Blocked
  - A task that is waiting for an event is said to be in the ‘Blocked’ state.
  - Two different types of event
    1. Temporal (time-related) events : Timer expiration
    2. Synchronization events : FreeRTOS queues, binary semaphores, counting semaphores, mutexes, recursive mutexes, event groups and direct to task notifications
- Suspended
  - Tasks in the Suspended state are not available to the scheduler.
  - Only way into the Suspended state is through a call to the vTaskSuspend()
  - Only way out being through a call to the vTaskResume() or xTaskResumeFromISR()
- Ready
  - Tasks that are in the Not Running state but are not Blocked or Suspended are said to be in the Ready state.
  - A new task is a ready one

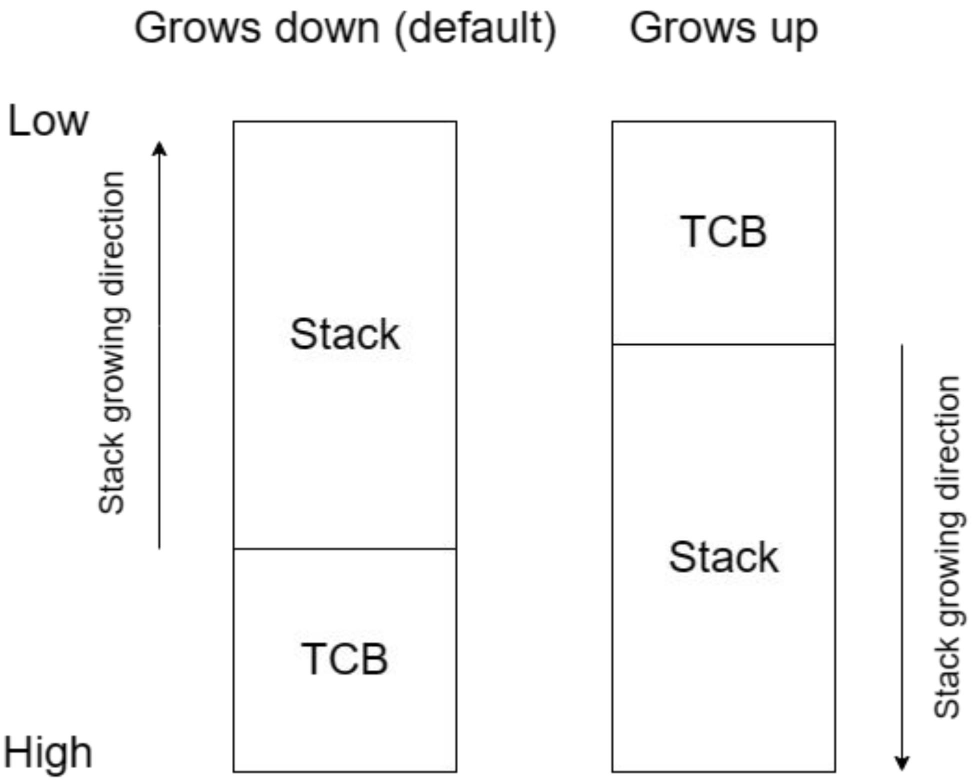
Task States



161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

Task Stack

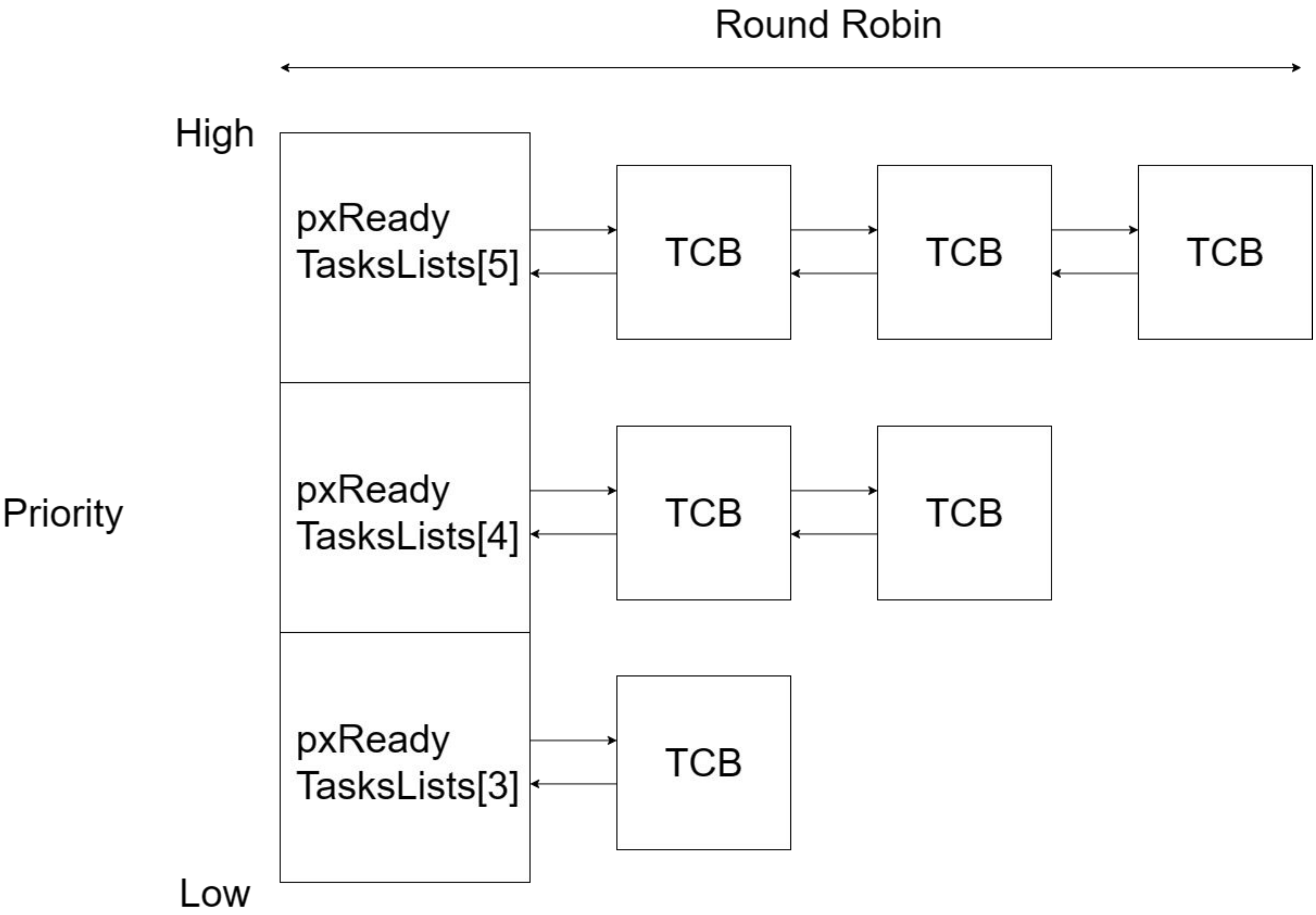
- ucHeap: a memory region for dynamic memory allocation, e.g., pvPortMalloc()
- xTaskCreate() calls pvPortMalloc() to allocate memory space for the stack and TCB of the new task
- memory layout



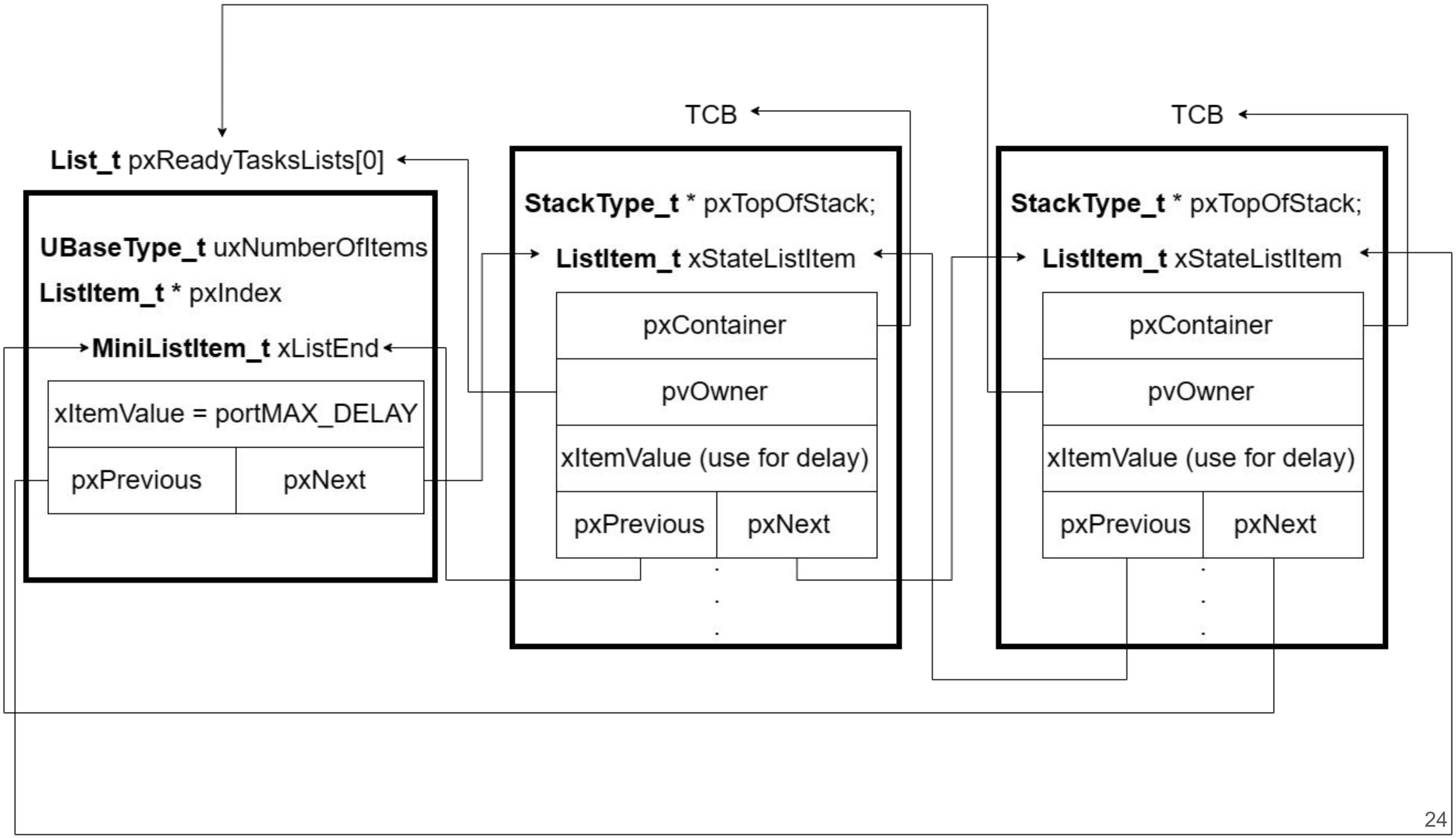
Task Ready List

- Task importance level is proportional to the priority value
    - 0=lowest priority
- Multiple tasks can have the same priority
    - Tasks of the same priority is scheduled using RR
    - Time slice on MSP430 is 1ms (configTICK_RATE_HZ = 1000 Hz)
- Multilevel Round Robin
    - The scheduler selects the highest, non-empty list
    - If there are multiple tasks on the list, RR will be used

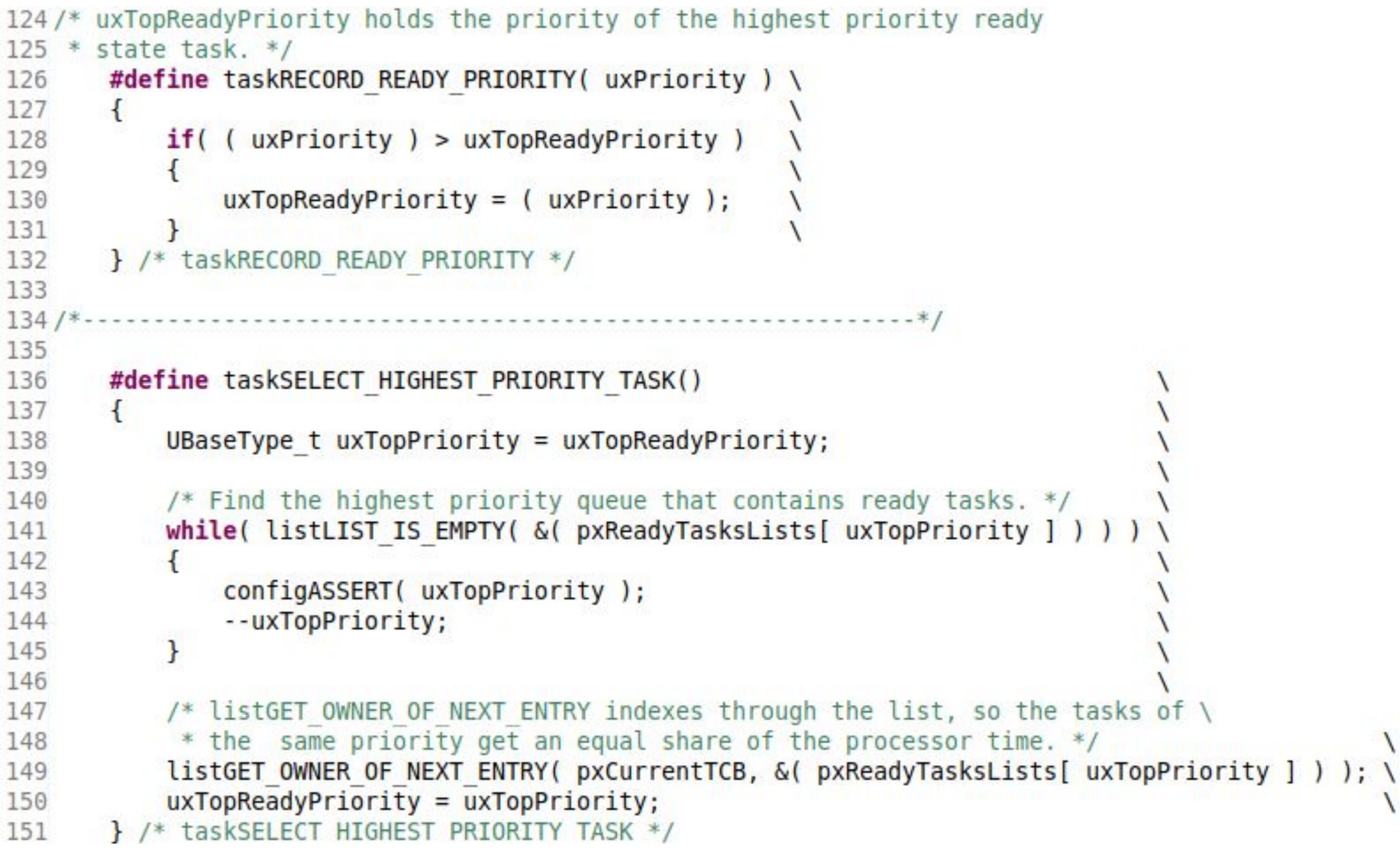
Task Ready List



Task Ready List


List_t pxReadyTasksLists[0]
UBaseType_t uxNumberOfItems
ListItem_t * pxIndex
MiniListItem_t xListEnd
xItemValue = portMAX_DELAY
pxPrevious
pxNext
TCB
StackType_t * pxTopOfStack;
ListItem_t xStateListItem
pxContainer
pvOwner
xItemValue (use for delay)
pxPrevious
pxNext
TCB
StackType_t * pxTopOfStack;
ListItem_t xStateListItem
pxContainer
pvOwner
xItemValue (use for delay)
pxPrevious
pxNext

Task Ready List



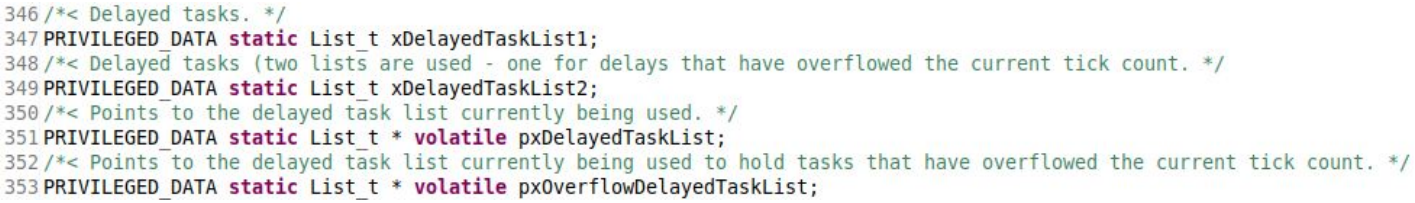
```
/* uxTopReadyPriority holds the priority of the highest priority ready
 * state task. */
    #define taskRECORD_READY_PRIORITY( uxPriority ) \
    {                                               \
        if( ( uxPriority ) > uxTopReadyPriority )   \
        {                                           \
            uxTopReadyPriority = ( uxPriority );    \
        }                                           \
    } /* taskRECORD_READY_PRIORITY */

/*-----------------------------------------------------------*/

    #define taskSELECT_HIGHEST_PRIORITY_TASK()                                \
    {                                                                         \
        UBaseType_t uxTopPriority = uxTopReadyPriority;                       \
                                                                              \
        /* Find the highest priority queue that contains ready tasks. */      \
        while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopPriority ] ) ) ) \
        {                                                                     \
            configASSERT( uxTopPriority );                                    \
            --uxTopPriority;                                                  \
        }                                                                     \
                                                                              \
        /* listGET_OWNER_OF_NEXT_ENTRY indexes through the list, so the tasks of \
         * the  same priority get an equal share of the processor time. */                    \
        listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists[ uxTopPriority ] ) ); \
        uxTopReadyPriority = uxTopPriority;                                                   \
    } /* taskSELECT_HIGHEST_PRIORITY_TASK */
```

Delay List

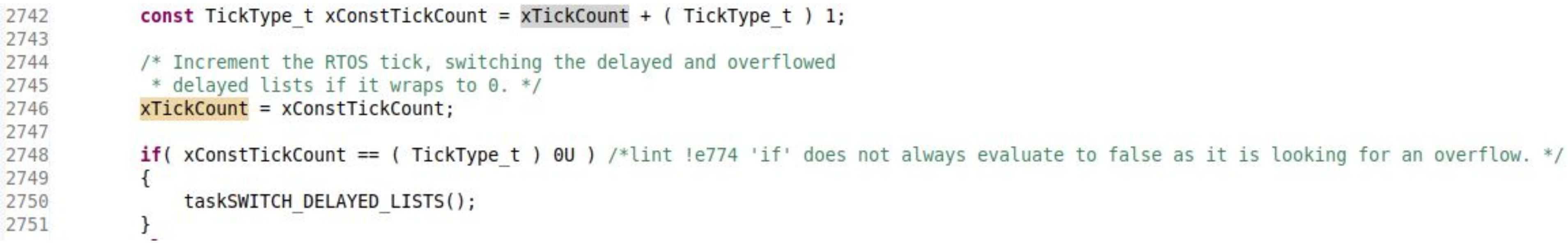
- Sleeping tasks are on this list
- Timer list is implemented as a **sorted** list
  - 100, 150, 80 → 80, 100, 150
- Different from uC/OS, the timer values are “absolute values”
- Counting up, not counting down

Delay List

- **xTickCount** is incremented by one on timer ISR
  - typedef uint32_t TickType_t
- Time slice=1ms and xTickCount=32 bit
  - Overflow occurs after about 49 days
- FreeRTOS uses two timer lists so that tick count can cycle through 2^32
  - If xTickCount becomes 0 after being incremented (+1) → overflow
  - Swap the two lists



```
/*< Delayed tasks. */
PRIVILEGED_DATA static List_t xDelayedTaskList1;
/*< Delayed tasks (two lists are used - one for delays that have overflowed the current tick count. */
PRIVILEGED_DATA static List_t xDelayedTaskList2;
/*< Points to the delayed task list currently being used. */
PRIVILEGED_DATA static List_t * volatile pxDelayedTaskList;
/*< Points to the delayed task list currently being used to hold tasks that have overflowed the current tick count. */
PRIVILEGED_DATA static List_t * volatile pxOverflowDelayedTaskList;
```



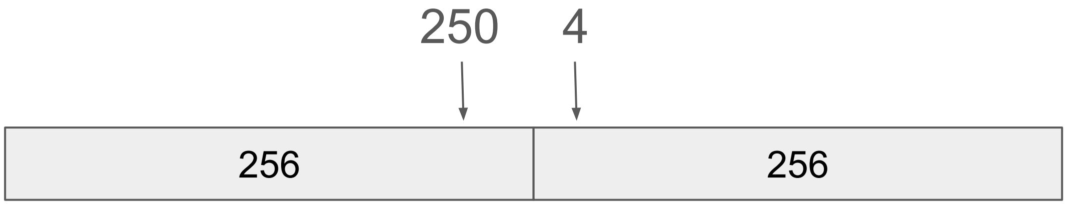
```
	const TickType_t xConstTickCount = xTickCount + ( TickType_t ) 1;

	/* Increment the RTOS tick, switching the delayed and overflowed
	 * delayed lists if it wraps to 0. */
	xTickCount = xConstTickCount;

	if( xConstTickCount == ( TickType_t ) 0U ) /*lint !e774 'if' does not always evaluate to false as it is looking for an overflow. */
	{
		taskSWITCH_DELAYED_LISTS();
	}
```

Swap lists

2^32

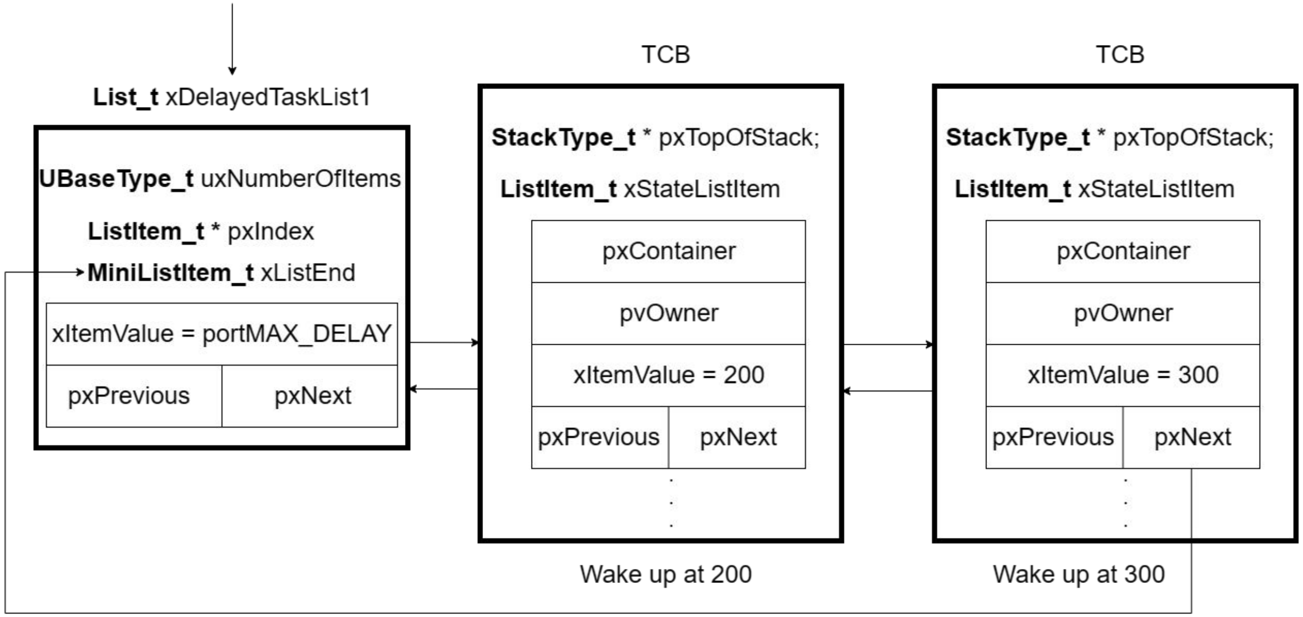
2^32

- For example, let the tick counter be 8 bits and let the current time is at 250
- If a delay is set to 10 ticks later, then it will be tick 4 in the overflow list



- In uCOS this does not happen because the timer list is “countdown”, not “countup”

pxDelayedTaskList



List_t xDelayedTaskList1

UBaseType_t uxNumberOfItems

ListItem_t * pxIndex

MiniListItem_t xListEnd

xItemValue = portMAX_DELAY

pxPrevious | pxNext

TCB

StackType_t * pxTopOfStack;

ListItem_t xStateListItem

pxContainer

pvOwner

xItemValue = 200

pxPrevious | pxNext

Wake up at 200

TCB

StackType_t * pxTopOfStack;

ListItem_t xStateListItem

pxContainer

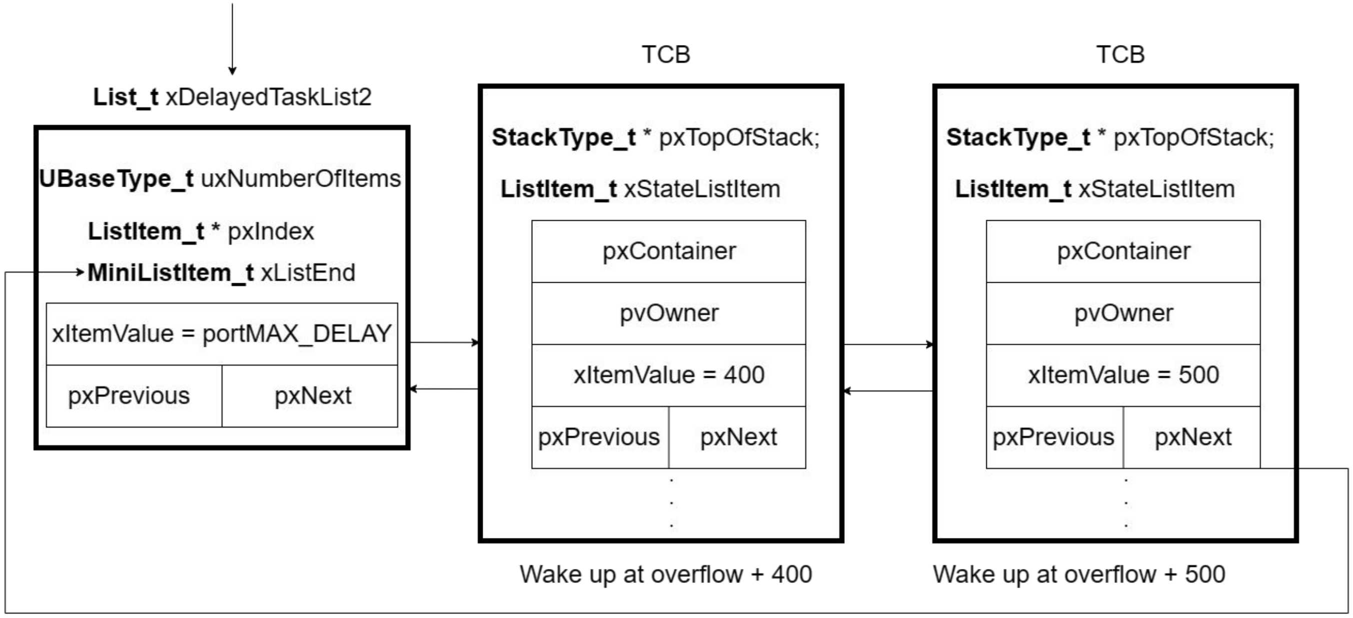
pvOwner

xItemValue = 300

pxPrevious | pxNext

Wake up at 300

pxOverflowDelayedTaskList



List_t xDelayedTaskList2

UBaseType_t uxNumberOfItems

ListItem_t * pxIndex

MiniListItem_t xListEnd

xItemValue = portMAX_DELAY

pxPrevious | pxNext

TCB

StackType_t * pxTopOfStack;

ListItem_t xStateListItem

pxContainer

pvOwner

xItemValue = 400

pxPrevious | pxNext

Wake up at overflow + 400

TCB

StackType_t * pxTopOfStack;

ListItem_t xStateListItem

pxContainer

pvOwner

xItemValue = 500

pxPrevious | pxNext

Wake up at overflow + 500

Essential Kernel Routines

- Kernel function naming convention
- Context Switch (vPortYield)
- Timer Tick ISR (vPortPreemptiveTickISR)
- Task Create (xTaskCreate)
- Task Delay (vTaskDelay)

Kernel Variable Naming Convention

Type	prefix
char	c
int16_t (short)	s
int32_t (long)	l
BaseType_t or other	x
unsigned	u
pointer	p
void	v

Example:
uint8_t (unsigned char) -> uc
pointer to char -> pc

Kernel API Naming Convention

- Type of return + filename + function name

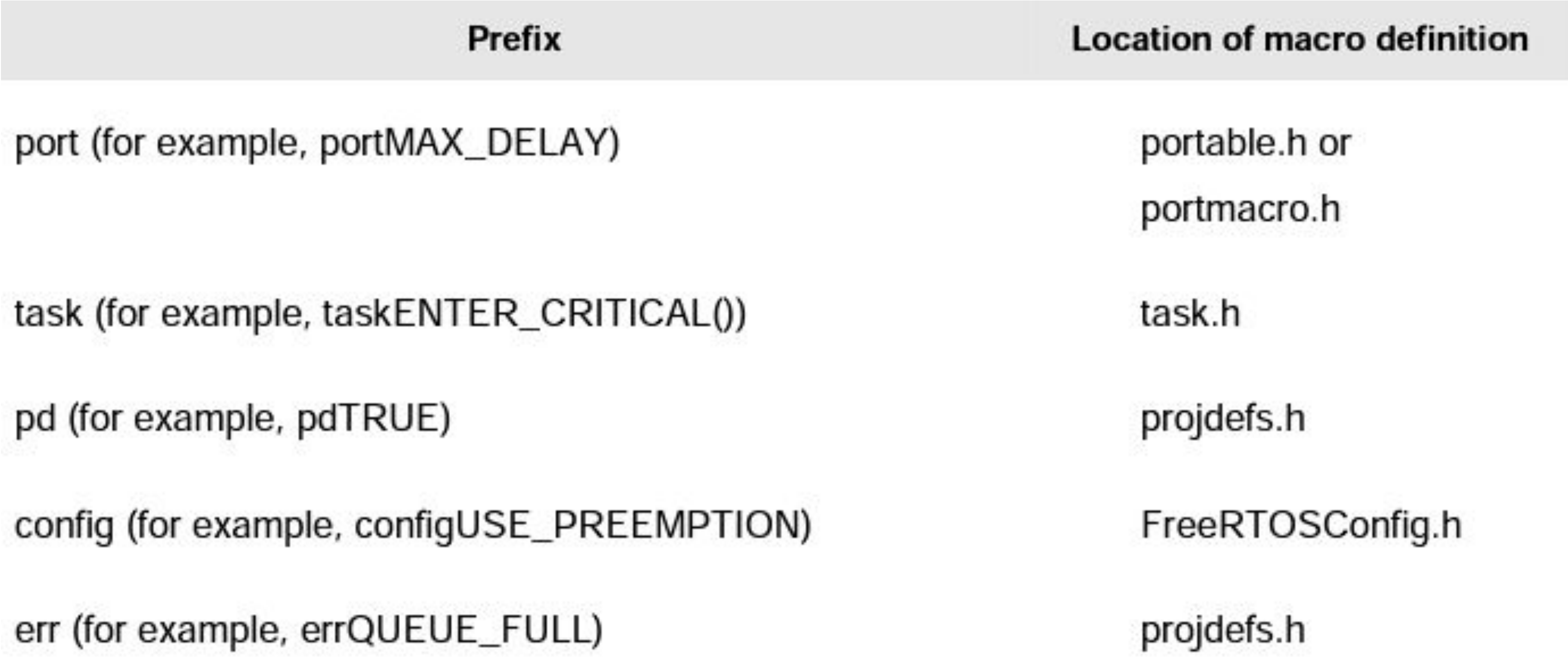
Example :

- xTaskCreate
  - return type = BaseType_t (x)
  - defined in task.c
  - Create()
- vTaskDelay
  - return type = void (v)
  - defined in task.c
  - Delay()
- pvTimerGetTimerID()
  - return type = pointer to void (pv)
  - defined in timer.c
  - GetTimerID()

File scope (private) functions have a prefix of ‘prv’.

Kernel Macro Naming Convention

Most macros are written in uppercase, and prefixed with lower case letters that indicate where the macro is defined.

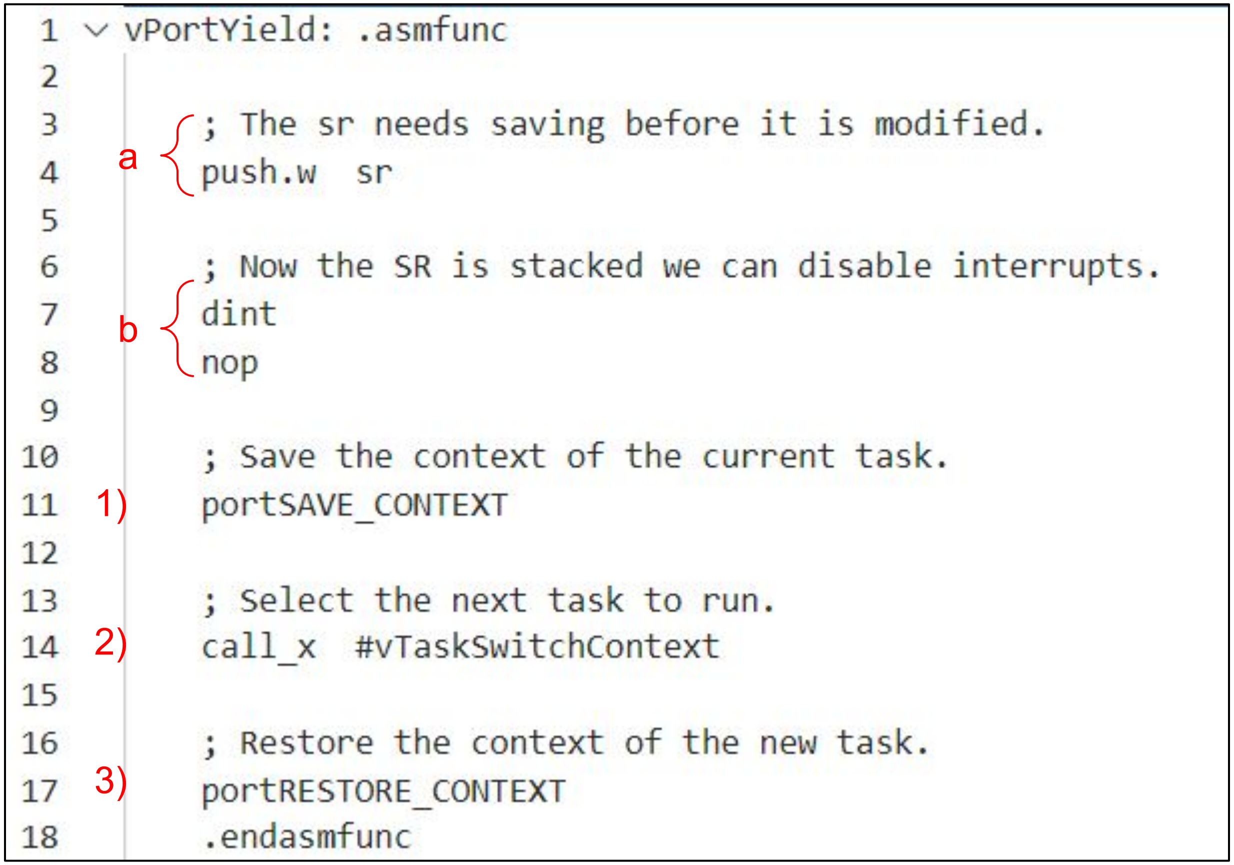


Prefix	Location of macro definition
port (for example, portMAX_DELAY)	portable.h or portmacro.h
task (for example, taskENTER_CRITICAL())	task.h
pd (for example, pdTRUE)	projdefs.h
config (for example, configUSE_PREEMPTION)	FreeRTOSConfig.h
err (for example, errQUEUE_FULL)	projdefs.h

Context Switch (Voluntarily Giving Up the CPU)

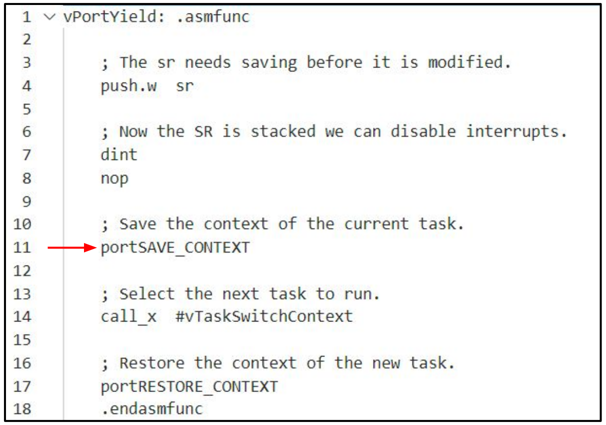
- vPortYield
  a. push sr into stack
  b. disable interrupt

Different from uCOS, “task-level” cxtsw do not trigger a software interrupt…



```
vPortYield: .asmfunc

    ; The sr needs saving before it is modified.
    push.w   sr

    ; Now the SR is stacked we can disable interrupts.
    dint
    nop

    ; Save the context of the current task.
    portSAVE_CONTEXT

    ; Select the next task to run.
    call_x   #vTaskSwitchContext

    ; Restore the context of the new task.
    portRESTORE_CONTEXT
    .endasmfunc
```

Context Switch

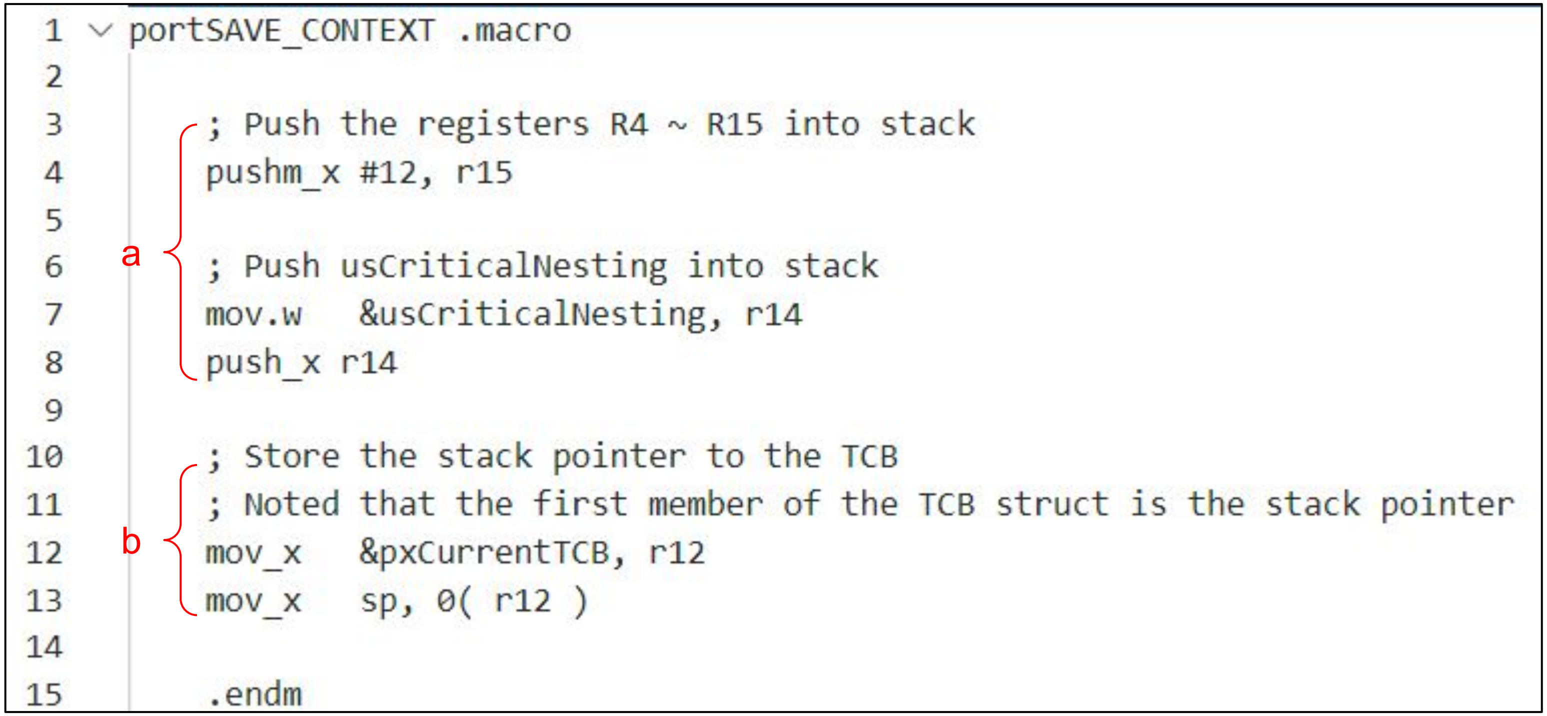


```
vPortYield: .asmfunc

    ; The sr needs saving before it is modified.
    push.w  sr

    ; Now the SR is stacked we can disable interrupts.
    dint
    nop

    ; Save the context of the current task.
    portSAVE_CONTEXT

    ; Select the next task to run.
    call_x  #vTaskSwitchContext

    ; Restore the context of the new task.
    portRESTORE_CONTEXT
    .endasmfunc
```

1) portSAVE_CONTEXT

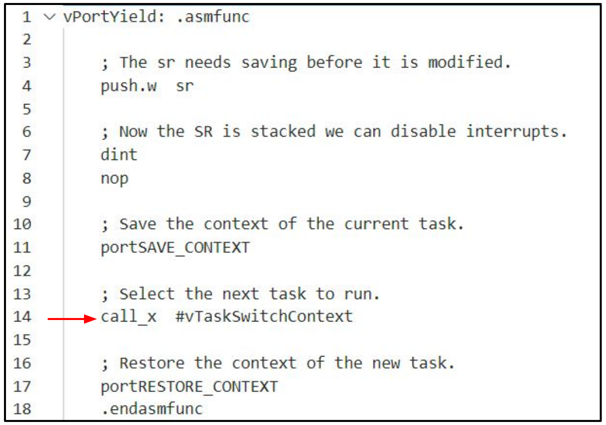
   a. push registers r4 ~ r15 and usCriticalNesting into stack

   b. store stack pointer to the TCB



```
portSAVE_CONTEXT .macro

    ; Push the registers R4 ~ R15 into stack
    pushm_x #12, r15

    ; Push usCriticalNesting into stack
    mov.w   &usCriticalNesting, r14
    push_x r14

    ; Store the stack pointer to the TCB
    ; Noted that the first member of the TCB struct is the stack pointer
    mov_x   &pxCurrentTCB, r12
    mov_x   sp, 0( r12 )

    .endm
```

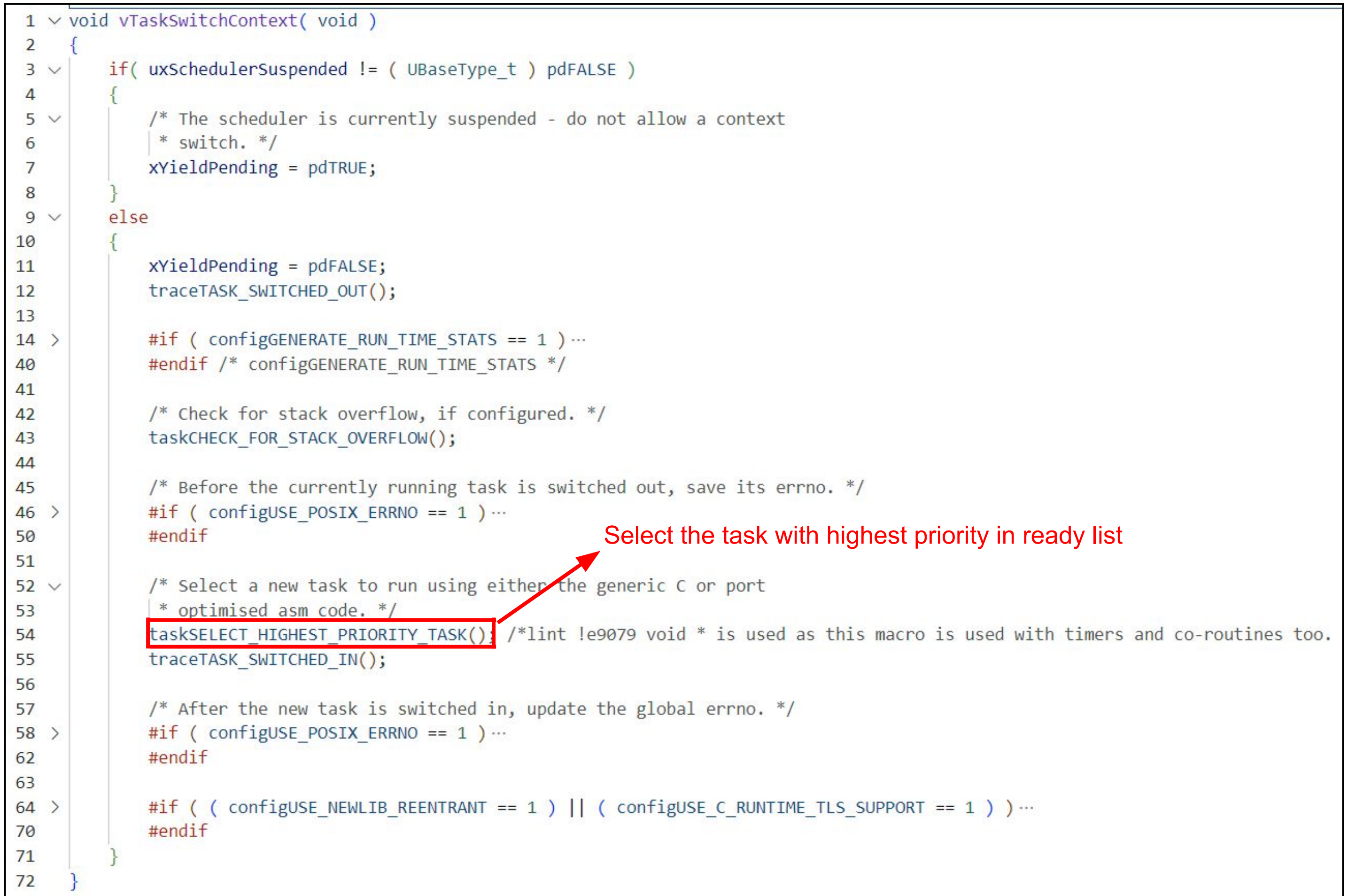
Context Switch



```
vPortYield: .asmfunc

    ; The sr needs saving before it is modified.
    push.w  sr

    ; Now the SR is stacked we can disable interrupts.
    dint
    nop

    ; Save the context of the current task.
    portSAVE_CONTEXT

    ; Select the next task to run.
    call_x  #vTaskSwitchContext

    ; Restore the context of the new task.
    portRESTORE_CONTEXT
    .endasmfunc
```

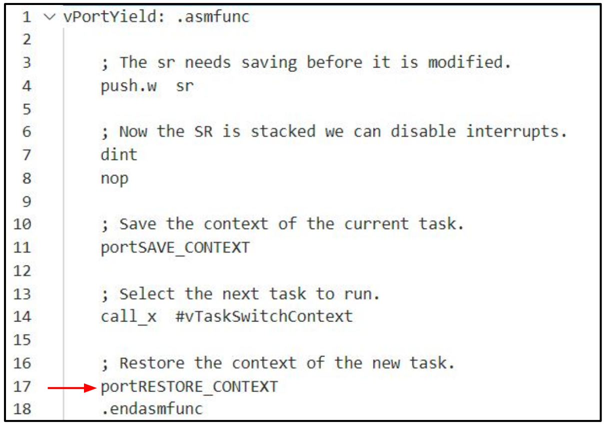
2) vTaskSwitchContext

   a. select the task to switch



```
void vTaskSwitchContext( void )
{
    if( uxSchedulerSuspended != ( UBaseType_t ) pdFALSE )
    {
        /* The scheduler is currently suspended - do not allow a context
         * switch. */
        xYieldPending = pdTRUE;
    }
    else
    {
        xYieldPending = pdFALSE;
        traceTASK_SWITCHED_OUT();

        #if ( configGENERATE_RUN_TIME_STATS == 1 )···
        #endif /* configGENERATE_RUN_TIME_STATS */

        /* Check for stack overflow, if configured. */
        taskCHECK_FOR_STACK_OVERFLOW();

        /* Before the currently running task is switched out, save its errno. */
        #if ( configUSE_POSIX_ERRNO == 1 )···
        #endif

        /* Select a new task to run using either the generic C or port
         * optimised asm code. */
        taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-routines too.
        traceTASK_SWITCHED_IN();

        /* After the new task is switched in, update the global errno. */
        #if ( configUSE_POSIX_ERRNO == 1 )···
        #endif

        #if ( ( configUSE_NEWLIB_REENTRANT == 1 ) || ( configUSE_C_RUNTIME_TLS_SUPPORT == 1 ) )···
        #endif
    }
}
```

Select the task with highest priority in ready list



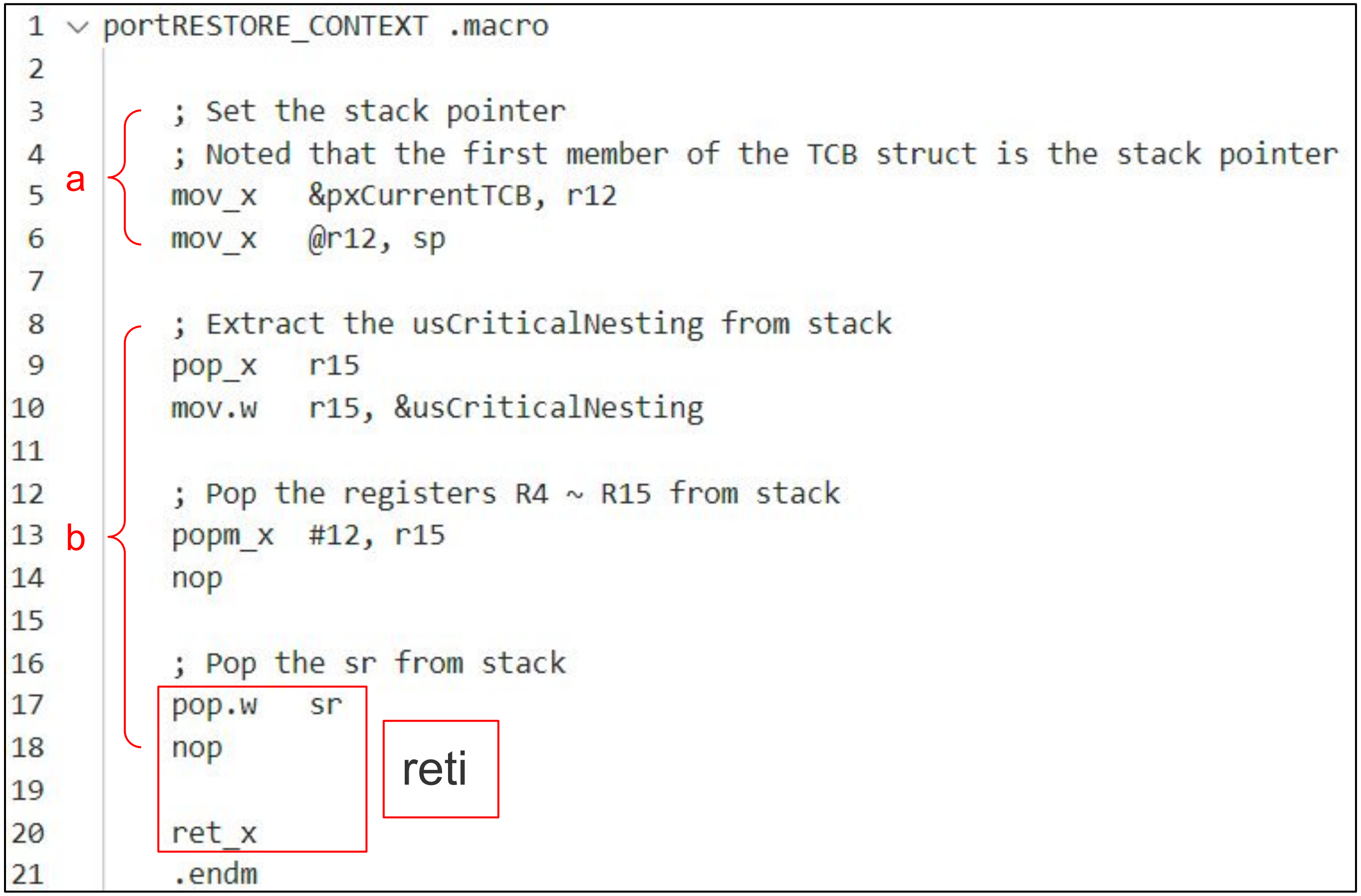
```
vPortYield: .asmfunc

    ; The sr needs saving before it is modified.
    push.w  sr

    ; Now the SR is stacked we can disable interrupts.
    dint
    nop

    ; Save the context of the current task.
    portSAVE_CONTEXT

    ; Select the next task to run.
    call_x  #vTaskSwitchContext

    ; Restore the context of the new task.
    portRESTORE_CONTEXT
    .endasmfunc
```

Context Switch

3) portRESTORE_CONTEXT

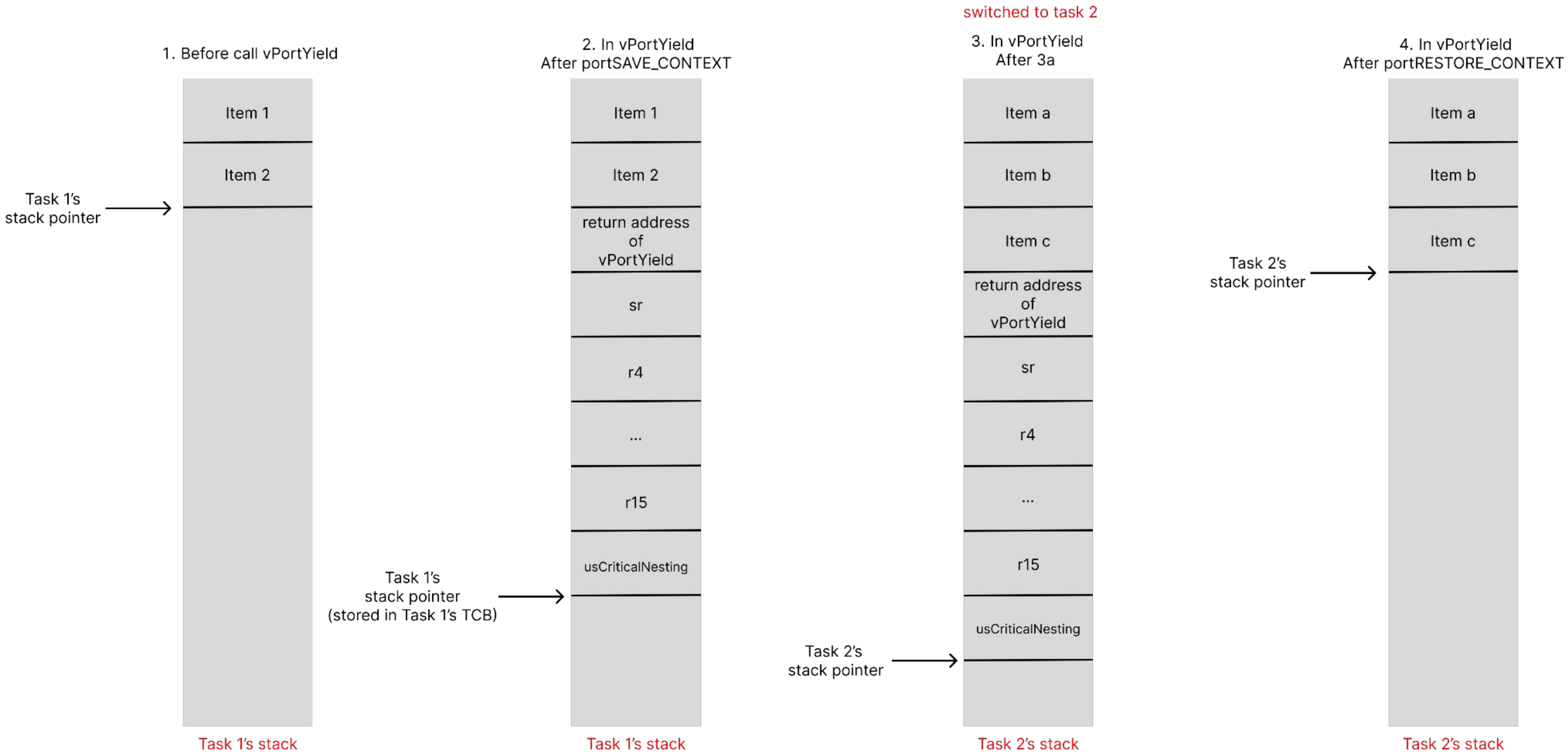
   a. set the stack pointer

   b. pop usCriticalNesting, registers r4 ~ r15 and sr from stack



```
portRESTORE_CONTEXT .macro

    ; Set the stack pointer
    ; Noted that the first member of the TCB struct is the stack pointer
    mov_x   &pxCurrentTCB, r12
    mov_x   @r12, sp

    ; Extract the usCriticalNesting from stack
    pop_x   r15
    mov.w   r15, &usCriticalNesting

    ; Pop the registers R4 ~ R15 from stack
    popm_x  #12, r15
    nop

    ; Pop the sr from stack
    pop.w   sr
    nop

    ret_x
    .endm
```

Context Switch



switched to task 2

1. Before call vPortYield

Item 1

Item 2

Task 1's stack pointer

Task 1's stack

2. In vPortYield
After portSAVE_CONTEXT

Item 1

Item 2

return address of vPortYield

sr

r4

...

r15

usCriticalNesting

Task 1's stack pointer (stored in Task 1's TCB)

Task 1's stack

3. In vPortYield
After 3a

Item a

Item b

Item c

return address of vPortYield

sr

r4

...

r15

usCriticalNesting

Task 2's stack pointer

Task 2's stack

4. In vPortYield
After portRESTORE_CONTEXT

Item a

Item b

Item c

Task 2's stack pointer

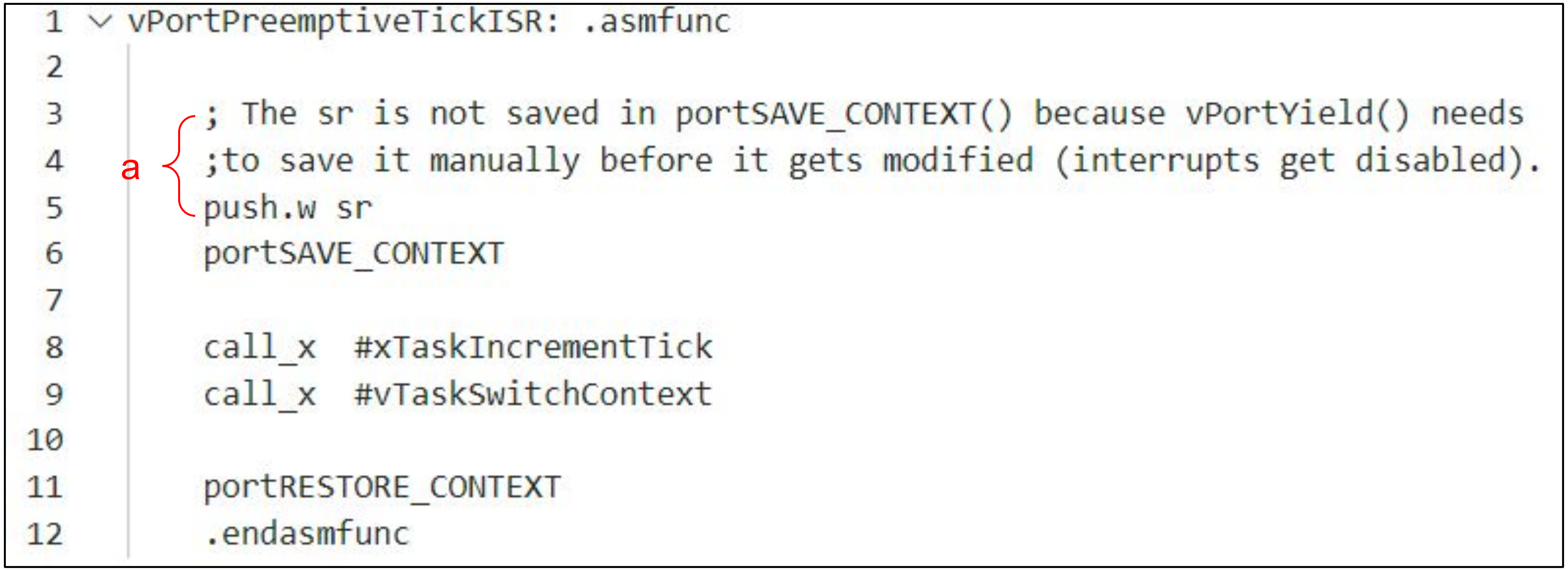
Task 2's stack

Timer Tick ISR

- Save CPU context
- Increment the time tick
  - Check tick overflow. If so, swap two timer lists
- If **xConstTickCount** is greater than **xNextTaskUnblockTime**, make the earliest waiting task ready
- Call context switch
- Restore CPU context and RETI
  - RETI = pop SR + ret

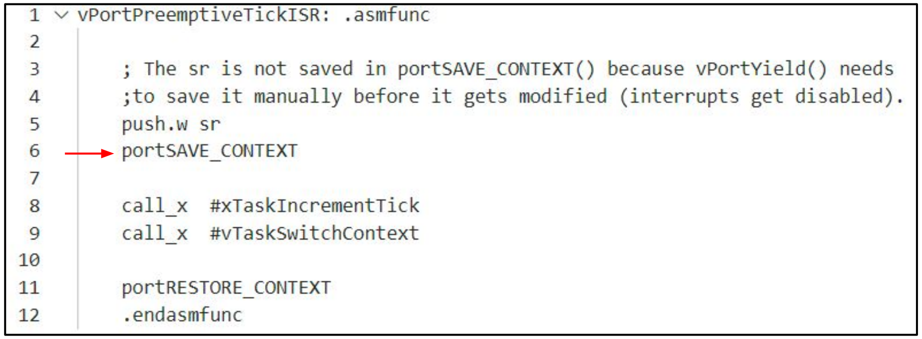
Timer Tick ISR

1. vPortPreemptiveTickISR
    a. push sr into stack



```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

Timer Tick ISR

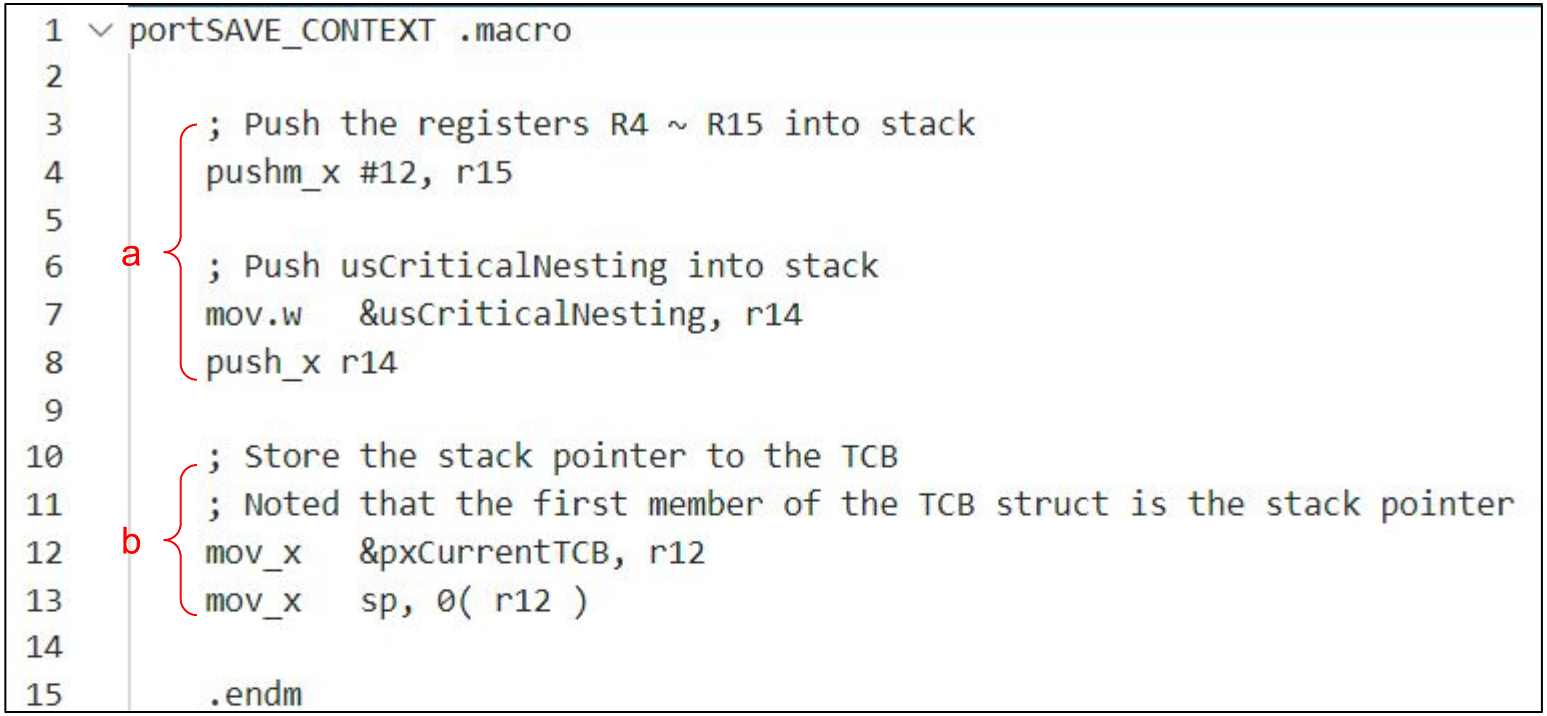


```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

2. portSAVE_CONTEXT

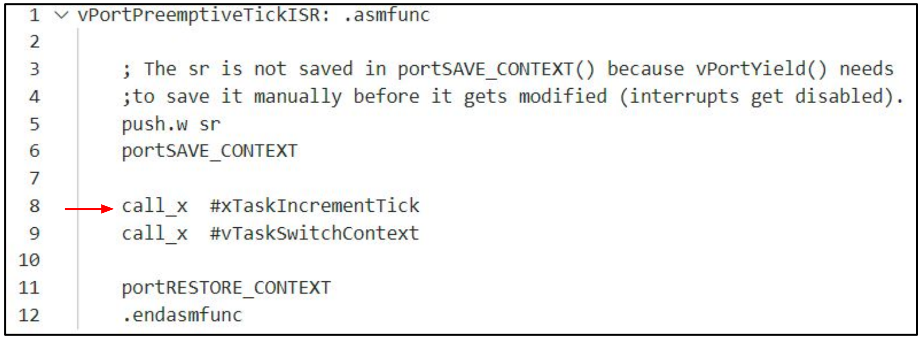
    a. push registers r4 ~ r15 and usCriticalNesting into stack

    b. store stack pointer to the TCB



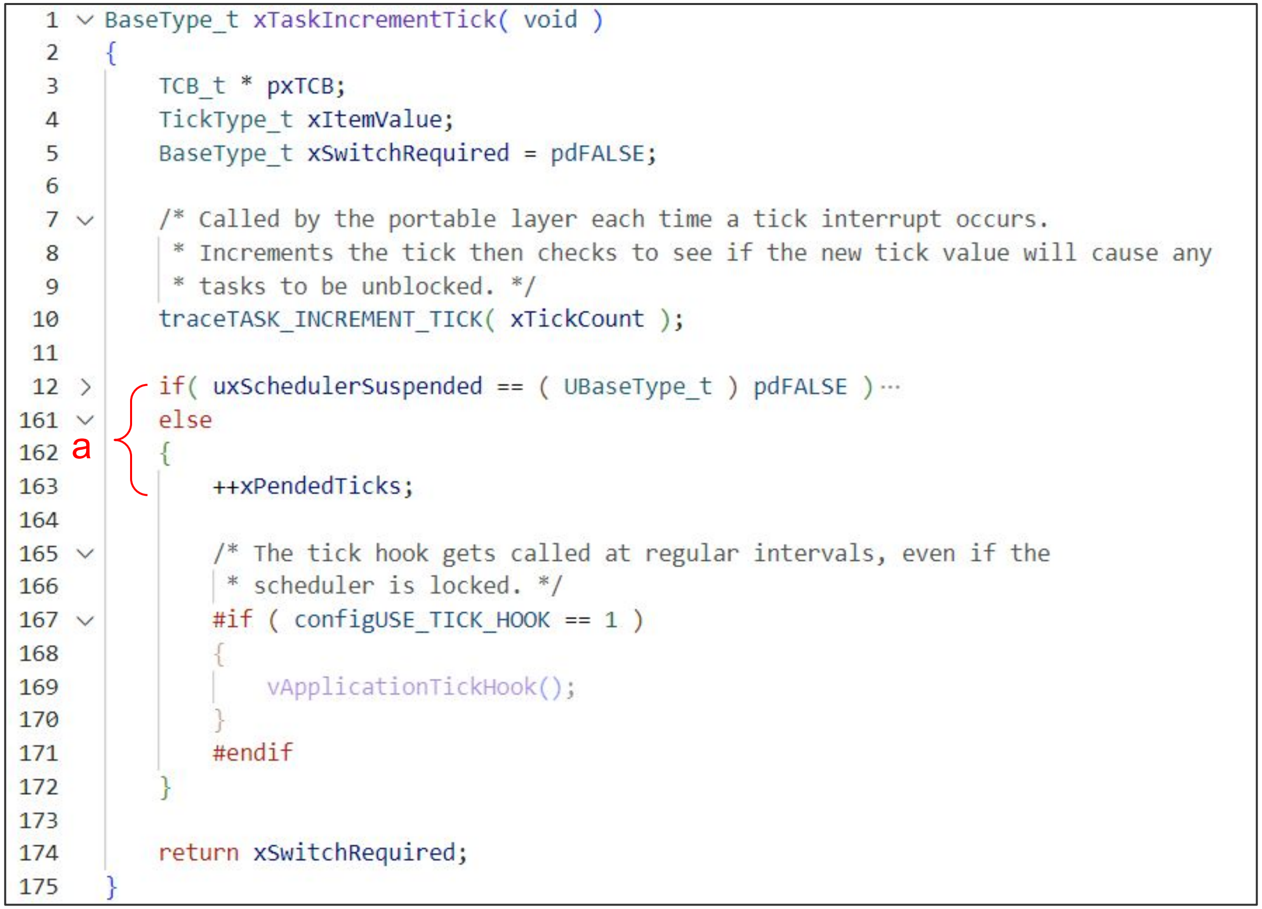
```
portSAVE_CONTEXT .macro

    ; Push the registers R4 ~ R15 into stack
    pushm_x #12, r15

    ; Push usCriticalNesting into stack
    mov.w   &usCriticalNesting, r14
    push_x r14

    ; Store the stack pointer to the TCB
    ; Noted that the first member of the TCB struct is the stack pointer
    mov_x   &pxCurrentTCB, r12
    mov_x   sp, 0( r12 )

    .endm
```

Timer Tick ISR



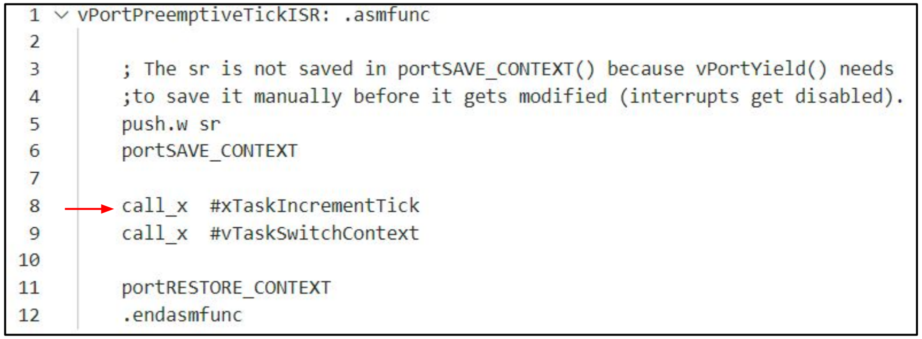
```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

3. xTaskIncrementTick
   a. Check if scheduler is suspended
   If scheduler is suspended, increase xPendedTicks by 1
   (so that we can handle the the accumulated ticks later)



```
BaseType_t xTaskIncrementTick( void )
{
    TCB_t * pxTCB;
    TickType_t xItemValue;
    BaseType_t xSwitchRequired = pdFALSE;

    /* Called by the portable layer each time a tick interrupt occurs.
     * Increments the tick then checks to see if the new tick value will cause any
     * tasks to be unblocked. */
    traceTASK_INCREMENT_TICK( xTickCount );

    if( uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )···
    else
    {
        ++xPendedTicks;

        /* The tick hook gets called at regular intervals, even if the
         * scheduler is locked. */
        #if ( configUSE_TICK_HOOK == 1 )
        {
            vApplicationTickHook();
        }
        #endif
    }

    return xSwitchRequired;
}
```

Timer Tick ISR

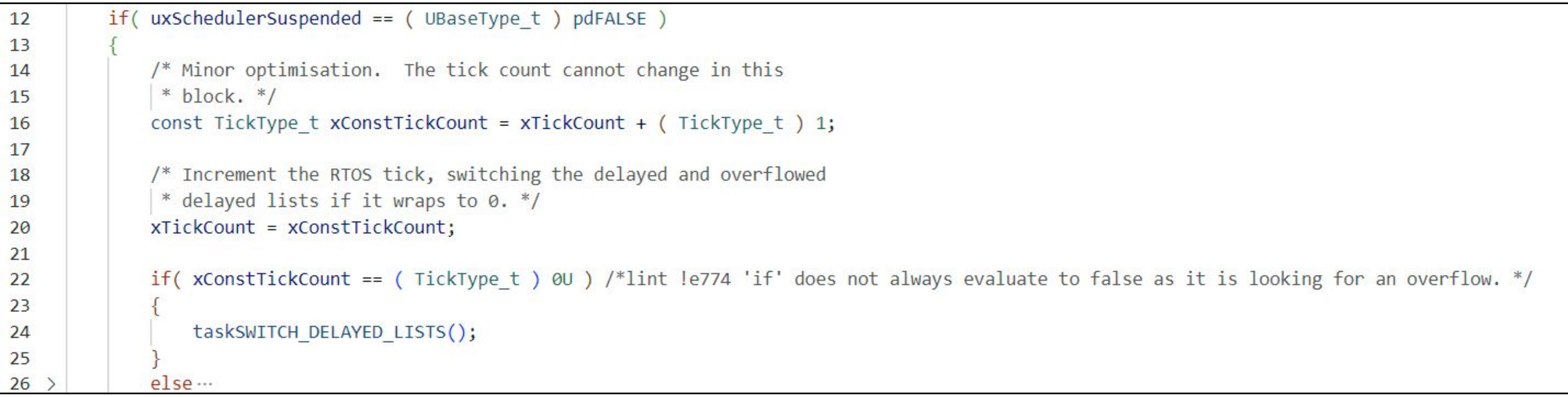


```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

3. xTaskIncrementTick

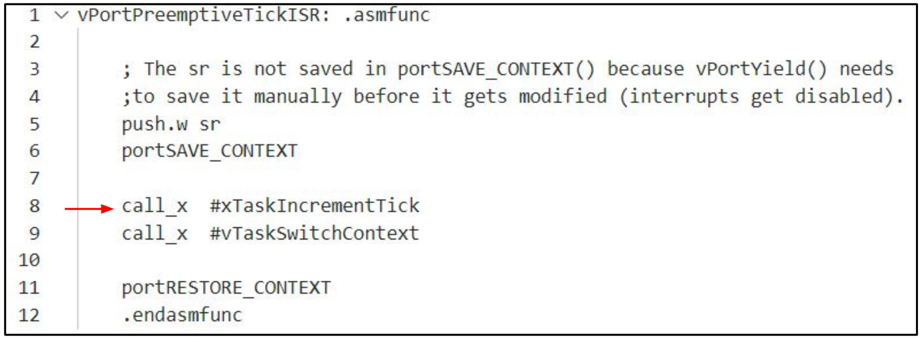
   b. Check if scheduler is suspended

   If scheduler is not suspended, increase xTickCount by 1 and check if it's overflowed



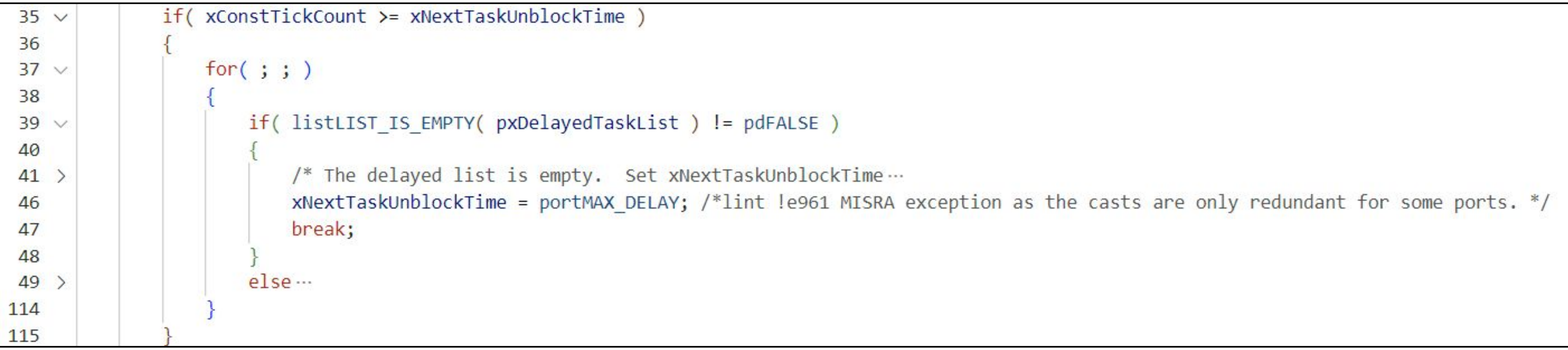
```
    if( uxSchedulerSuspended == ( UBaseType_t ) pdFALSE )
    {
        /* Minor optimisation.  The tick count cannot change in this
         * block. */
        const TickType_t xConstTickCount = xTickCount + ( TickType_t ) 1;

        /* Increment the RTOS tick, switching the delayed and overflowed
         * delayed lists if it wraps to 0. */
        xTickCount = xConstTickCount;

        if( xConstTickCount == ( TickType_t ) 0U ) /*lint !e774 'if' does not always evaluate to false as it is looking for an overflow. */
        {
            taskSWITCH_DELAYED_LISTS();
        }
        else···
```

Timer Tick ISR



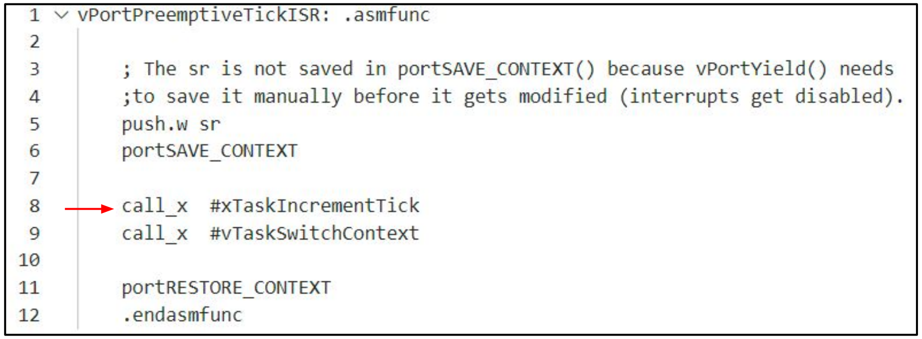
```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

3. xTaskIncrementTick
   c. check if this tick has made a timer (a sleeping task) expire
      and if the delayed list is empty, set next timeout to a big value



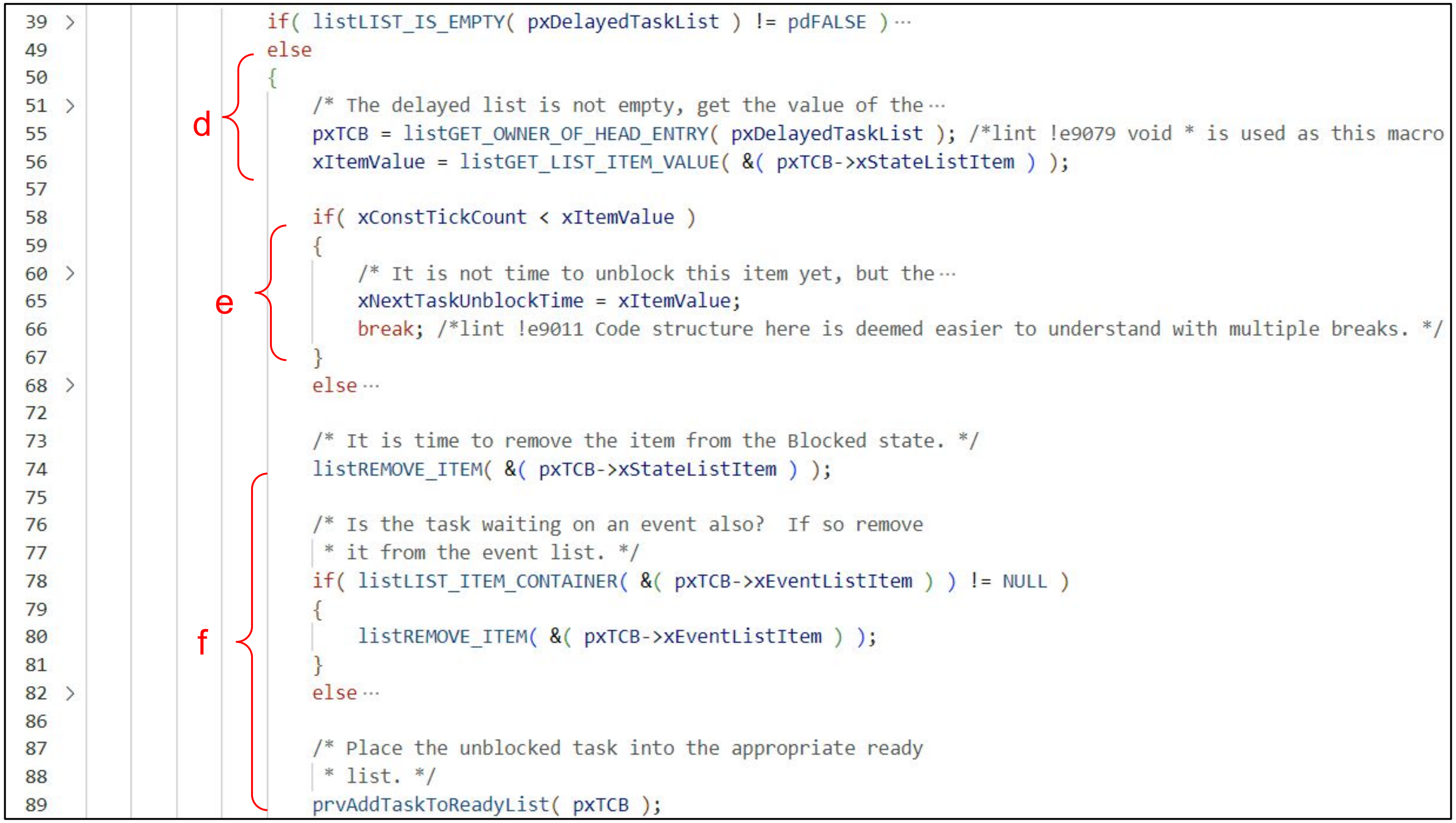
```
if( xConstTickCount >= xNextTaskUnblockTime )
{
    for( ; ; )
    {
        if( listLIST_IS_EMPTY( pxDelayedTaskList ) != pdFALSE )
        {
            /* The delayed list is empty.  Set xNextTaskUnblockTime···
            xNextTaskUnblockTime = portMAX_DELAY; /*lint !e961 MISRA exception as the casts are only redundant for some ports. */
            break;
        }
        else···
    }
}
```

Timer Tick ISR



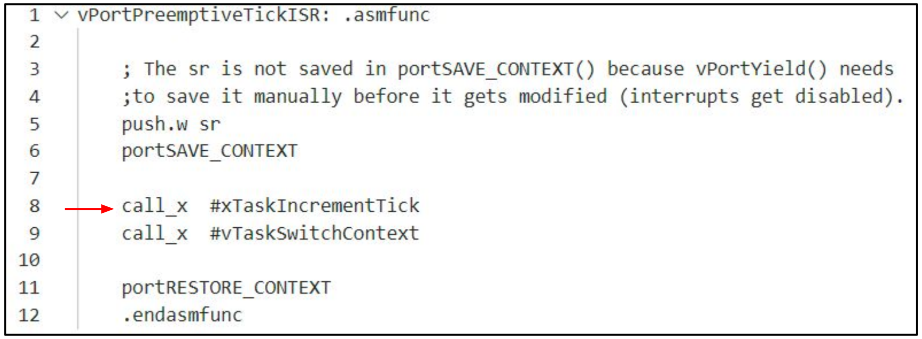
```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

3. xTaskIncrementTick
    d. if delayed list is not empty, extract the timeout of the task at the head of delayed list
    e. if it’s not yet to unblock the task, set next timeout
    f. if it’s time to unblock the task, remove it from delayed list and event list, then add it to ready list



```
if( listLIST_IS_EMPTY( pxDelayedTaskList ) != pdFALSE )⋯
else
{
    /* The delayed list is not empty, get the value of the⋯
    pxTCB = listGET_OWNER_OF_HEAD_ENTRY( pxDelayedTaskList ); /*lint !e9079 void * is used as this macro
    xItemValue = listGET_LIST_ITEM_VALUE( &( pxTCB->xStateListItem ) );

    if( xConstTickCount < xItemValue )
    {
        /* It is not time to unblock this item yet, but the⋯
        xNextTaskUnblockTime = xItemValue;
        break; /*lint !e9011 Code structure here is deemed easier to understand with multiple breaks. */
    }
    else⋯

    /* It is time to remove the item from the Blocked state. */
    listREMOVE_ITEM( &( pxTCB->xStateListItem ) );

    /* Is the task waiting on an event also?  If so remove
     * it from the event list. */
    if( listLIST_ITEM_CONTAINER( &( pxTCB->xEventListItem ) ) != NULL )
    {
        listREMOVE_ITEM( &( pxTCB->xEventListItem ) );
    }
    else⋯

    /* Place the unblocked task into the appropriate ready
     * list. */
    prvAddTaskToReadyList( pxTCB );
```

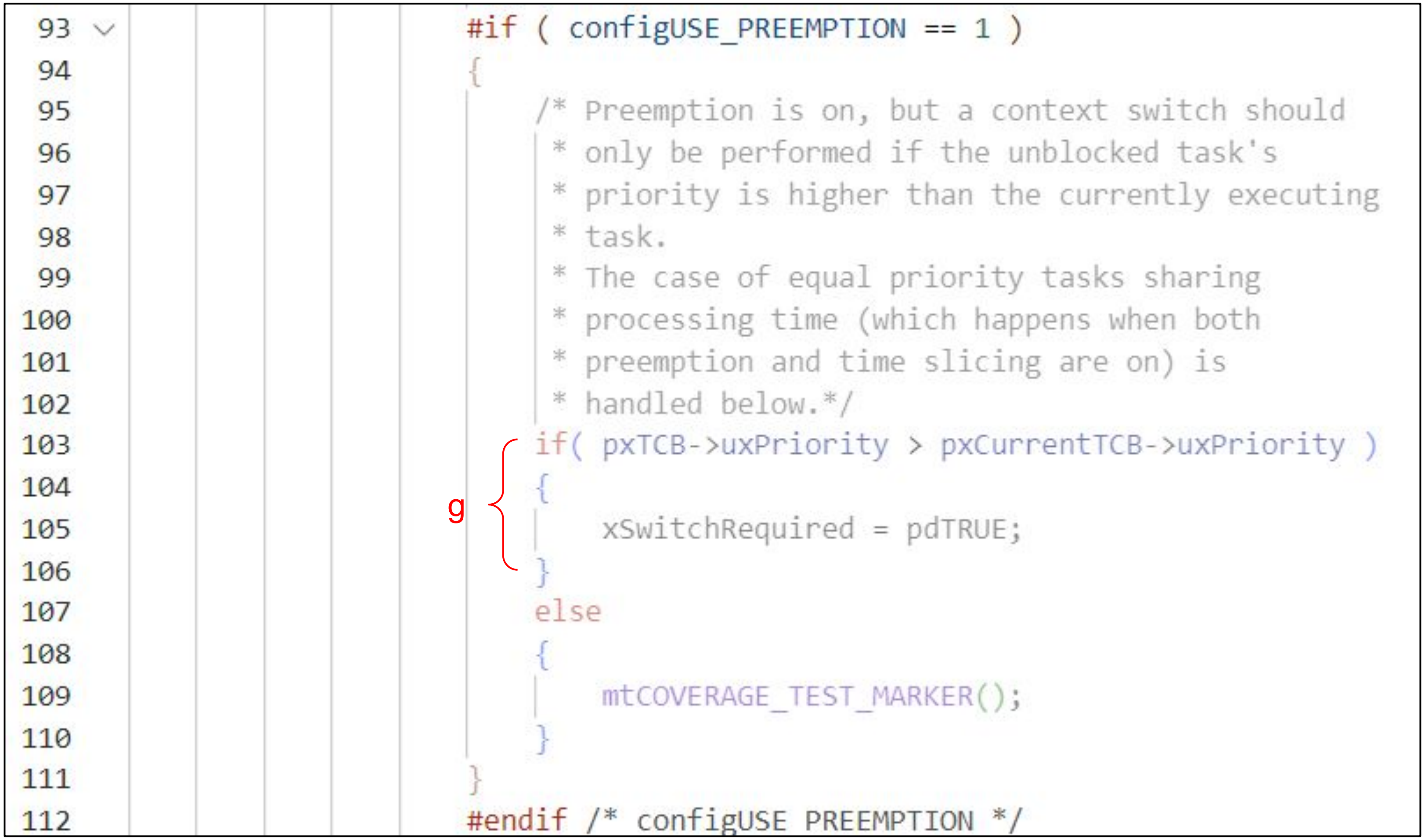
Timer Tick ISR



```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

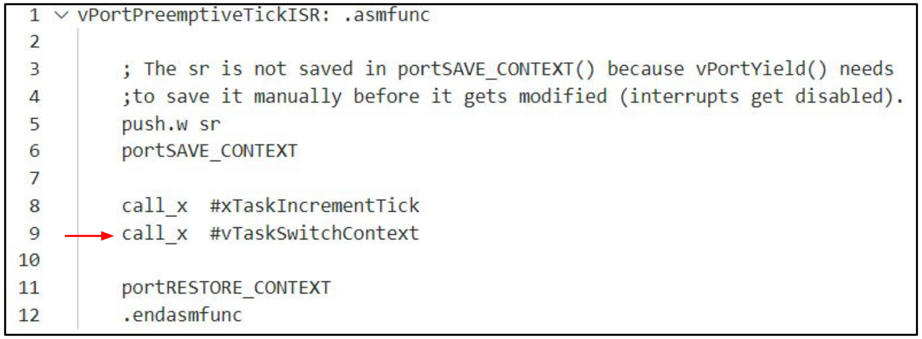
3. xTaskIncrementTick

   g. check if the task has higher priority than current task, if so, context switch is required



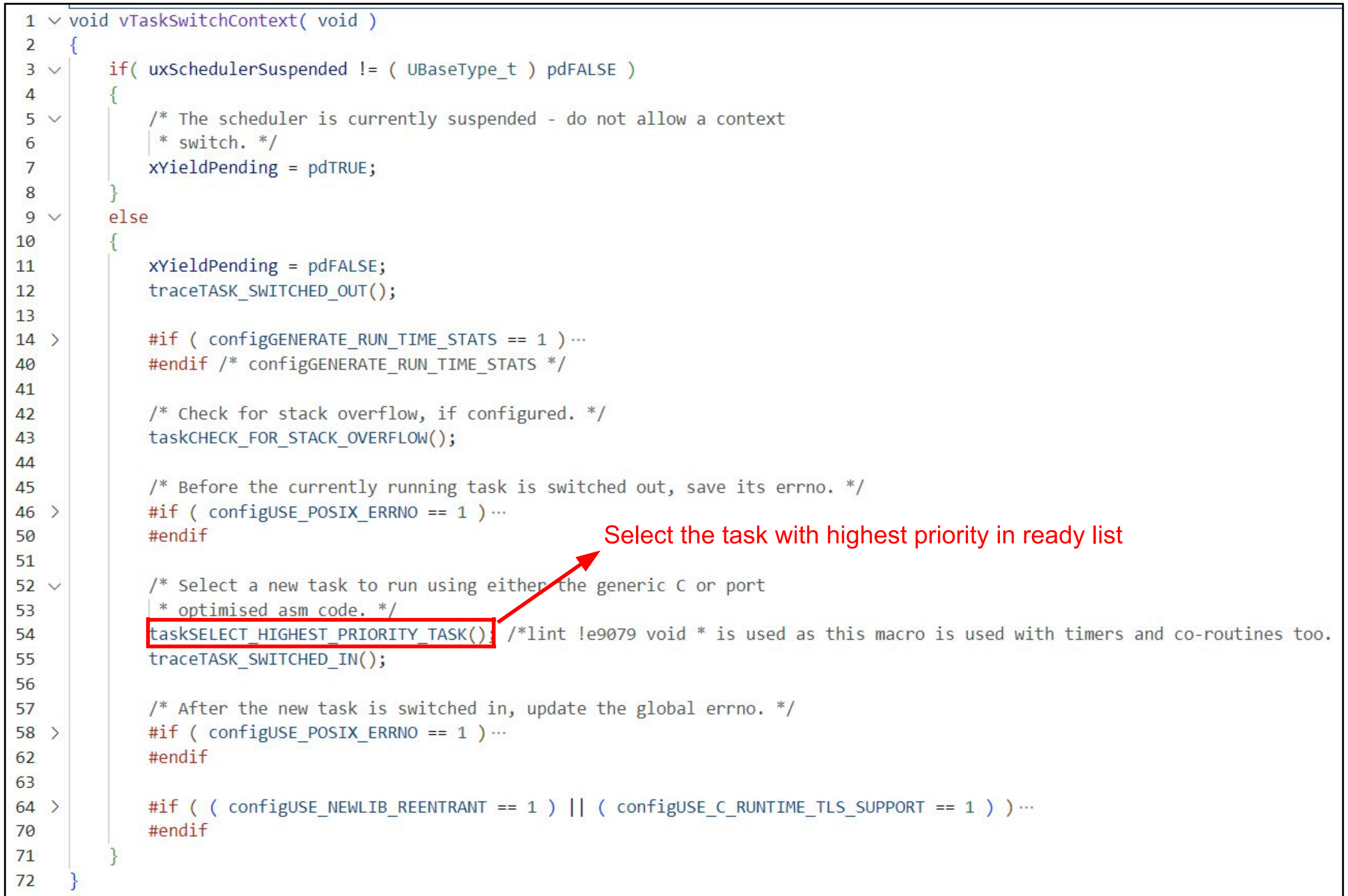
```
#if ( configUSE_PREEMPTION == 1 )
{
    /* Preemption is on, but a context switch should
     * only be performed if the unblocked task's
     * priority is higher than the currently executing
     * task.
     * The case of equal priority tasks sharing
     * processing time (which happens when both
     * preemption and time slicing are on) is
     * handled below.*/
    if( pxTCB->uxPriority > pxCurrentTCB->uxPriority )
    {
        xSwitchRequired = pdTRUE;
    }
    else
    {
        mtCOVERAGE_TEST_MARKER();
    }
}
#endif /* configUSE PREEMPTION */
```

Timer Tick ISR



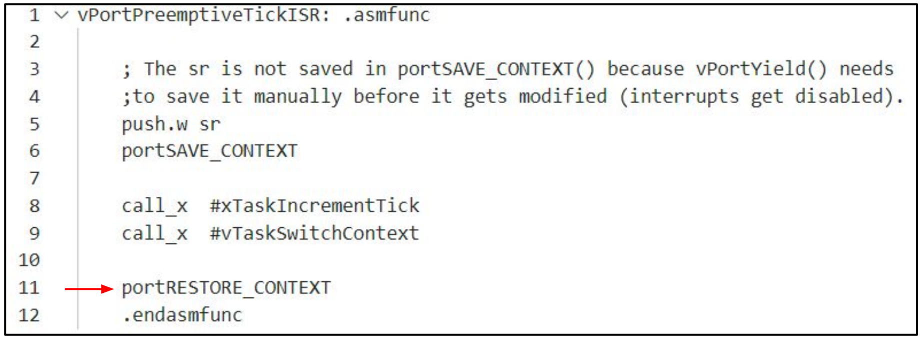
```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

4. vTaskSwitchContext
   a. select the task to switch



```
void vTaskSwitchContext( void )
{
    if( uxSchedulerSuspended != ( UBaseType_t ) pdFALSE )
    {
        /* The scheduler is currently suspended - do not allow a context
         * switch. */
        xYieldPending = pdTRUE;
    }
    else
    {
        xYieldPending = pdFALSE;
        traceTASK_SWITCHED_OUT();

        #if ( configGENERATE_RUN_TIME_STATS == 1 )⋯
        #endif /* configGENERATE_RUN_TIME_STATS */

        /* Check for stack overflow, if configured. */
        taskCHECK_FOR_STACK_OVERFLOW();

        /* Before the currently running task is switched out, save its errno. */
        #if ( configUSE_POSIX_ERRNO == 1 )⋯
        #endif

        /* Select a new task to run using either the generic C or port
         * optimised asm code. */
        taskSELECT_HIGHEST_PRIORITY_TASK(); /*lint !e9079 void * is used as this macro is used with timers and co-routines too.
        traceTASK_SWITCHED_IN();

        /* After the new task is switched in, update the global errno. */
        #if ( configUSE_POSIX_ERRNO == 1 )⋯
        #endif

        #if ( ( configUSE_NEWLIB_REENTRANT == 1 ) || ( configUSE_C_RUNTIME_TLS_SUPPORT == 1 ) )⋯
        #endif
    }
}
```

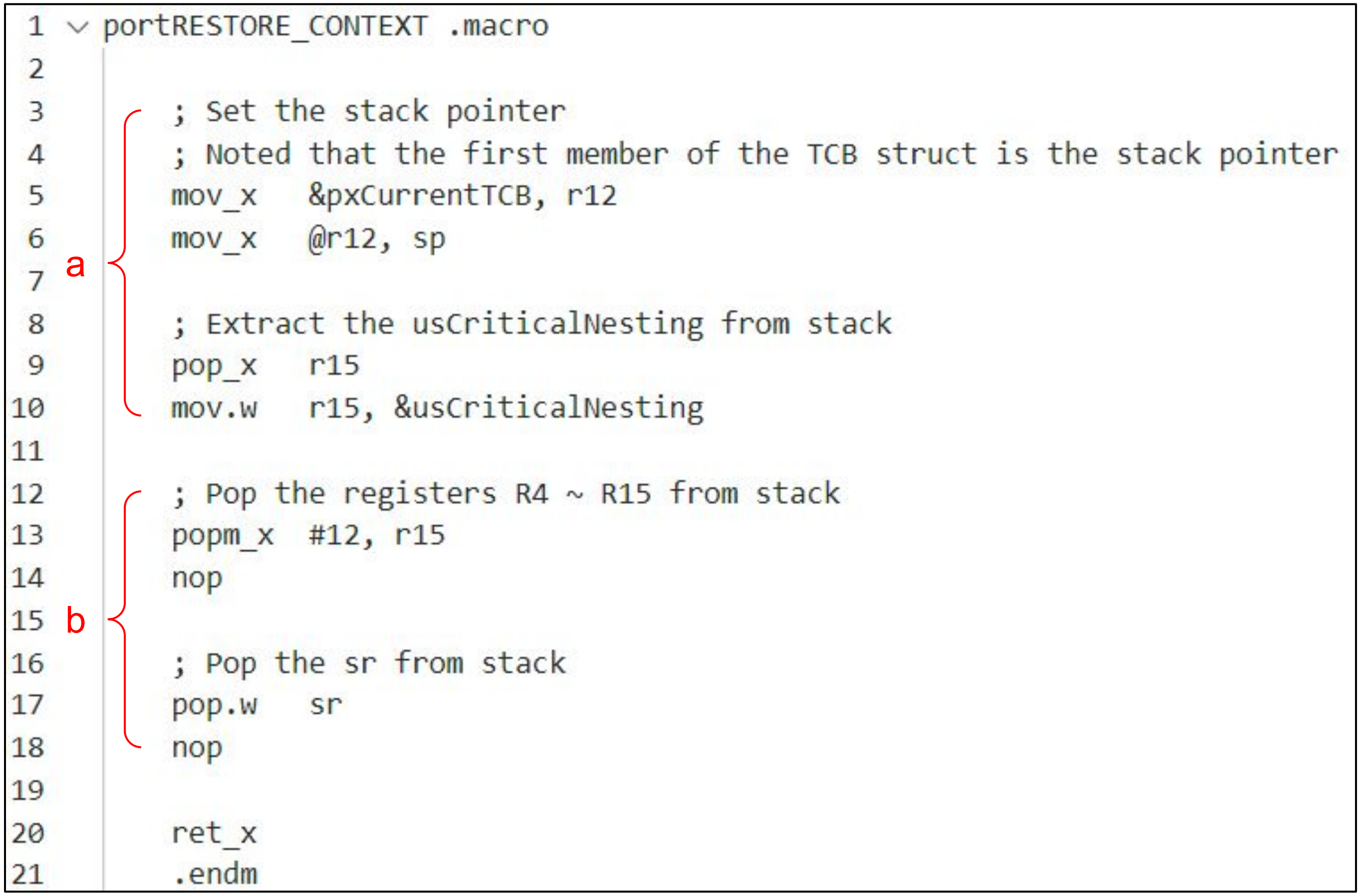
Select the task with highest priority in ready list



```
vPortPreemptiveTickISR: .asmfunc

    ; The sr is not saved in portSAVE_CONTEXT() because vPortYield() needs
    ;to save it manually before it gets modified (interrupts get disabled).
    push.w sr
    portSAVE_CONTEXT

    call_x  #xTaskIncrementTick
    call_x  #vTaskSwitchContext

    portRESTORE_CONTEXT
    .endasmfunc
```

Timer Tick ISR

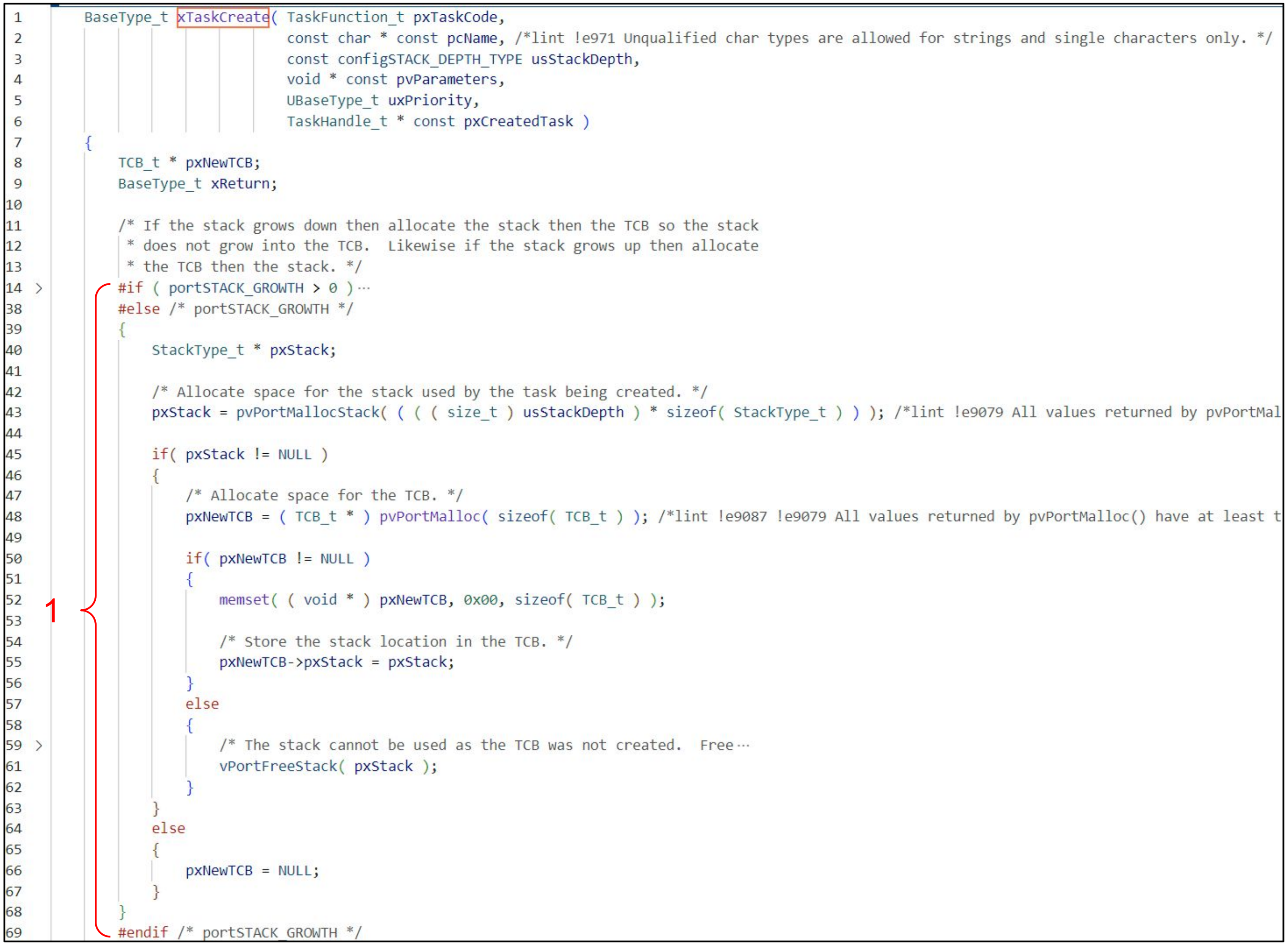
5. portRESTORE_CONTEXT
   a. set the stack pointer
   b. pop usCriticalNesting, registers r4 ~ r15 and sr from stack



```
portRESTORE_CONTEXT .macro

    ; Set the stack pointer
    ; Noted that the first member of the TCB struct is the stack pointer
    mov_x   &pxCurrentTCB, r12
    mov_x   @r12, sp

    ; Extract the usCriticalNesting from stack
    pop_x   r15
    mov.w   r15, &usCriticalNesting

    ; Pop the registers R4 ~ R15 from stack
    popm_x  #12, r15
    nop

    ; Pop the sr from stack
    pop.w   sr
    nop

    ret_x
    .endm
```

xTaskCreate()

1. Allocate space for stack and TCB



```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,
                        const char * const pcName, /*lint !e971 Unqualified char types are allowed for strings and single characters only. */
                        const configSTACK_DEPTH_TYPE usStackDepth,
                        void * const pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * const pxCreatedTask )
{
    TCB_t * pxNewTCB;
    BaseType_t xReturn;

    /* If the stack grows down then allocate the stack then the TCB so the stack
     * does not grow into the TCB.  Likewise if the stack grows up then allocate
     * the TCB then the stack. */
    #if ( portSTACK_GROWTH > 0 )⋯
    #else /* portSTACK_GROWTH */
    {
        StackType_t * pxStack;

        /* Allocate space for the stack used by the task being created. */
        pxStack = pvPortMallocStack( ( ( ( size_t ) usStackDepth ) * sizeof( StackType_t ) ) ); /*lint !e9079 All values returned by pvPortMal

        if( pxStack != NULL )
        {
            /* Allocate space for the TCB. */
            pxNewTCB = ( TCB_t * ) pvPortMalloc( sizeof( TCB_t ) ); /*lint !e9087 !e9079 All values returned by pvPortMalloc() have at least t

            if( pxNewTCB != NULL )
            {
                memset( ( void * ) pxNewTCB, 0x00, sizeof( TCB_t ) );

                /* Store the stack location in the TCB. */
                pxNewTCB->pxStack = pxStack;
            }
            else
            {
                /* The stack cannot be used as the TCB was not created.  Free⋯
                vPortFreeStack( pxStack );
            }
        }
        else
        {
            pxNewTCB = NULL;
        }
    }
    #endif /* portSTACK_GROWTH */
```

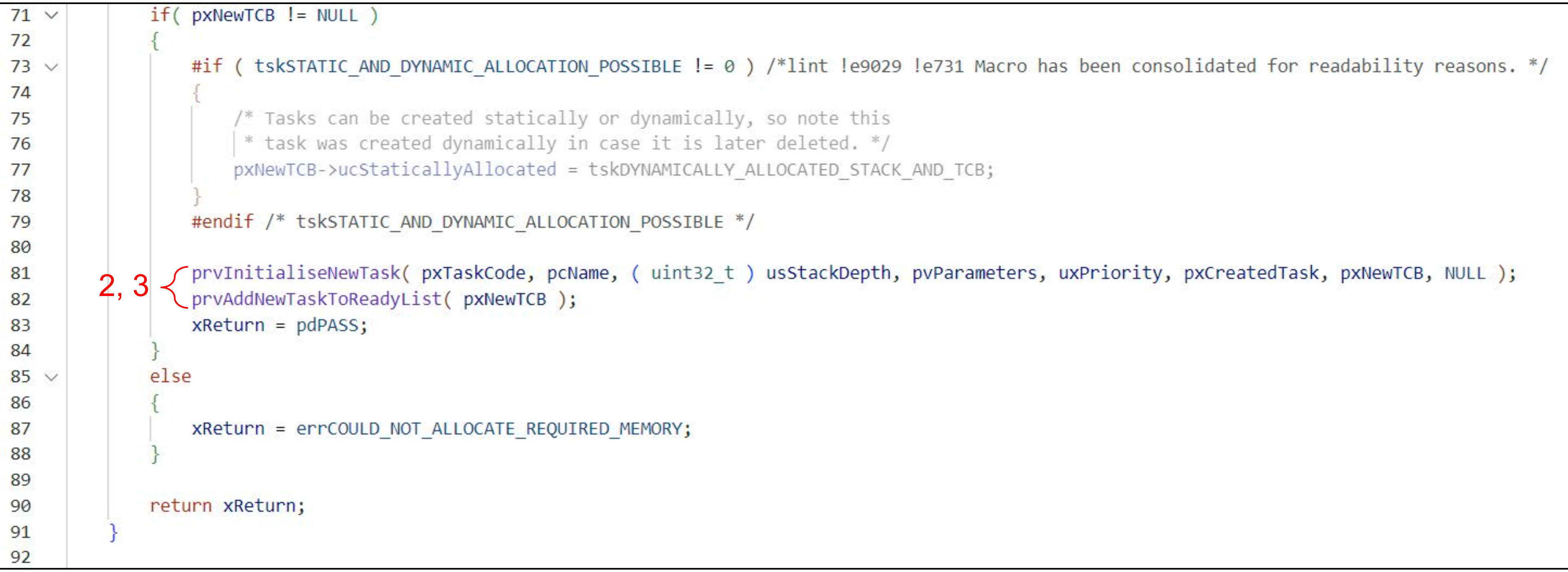
xTaskCreate()

2. call prvInitialiseNewTask to initialise the task:

   Initialise stack, assign task name, assign priority to the task's TCB

3. call prvAddNewTasktoReadyList adding the task to ready list:

   Make new task the current task if ready list is empty or new task has higher priority

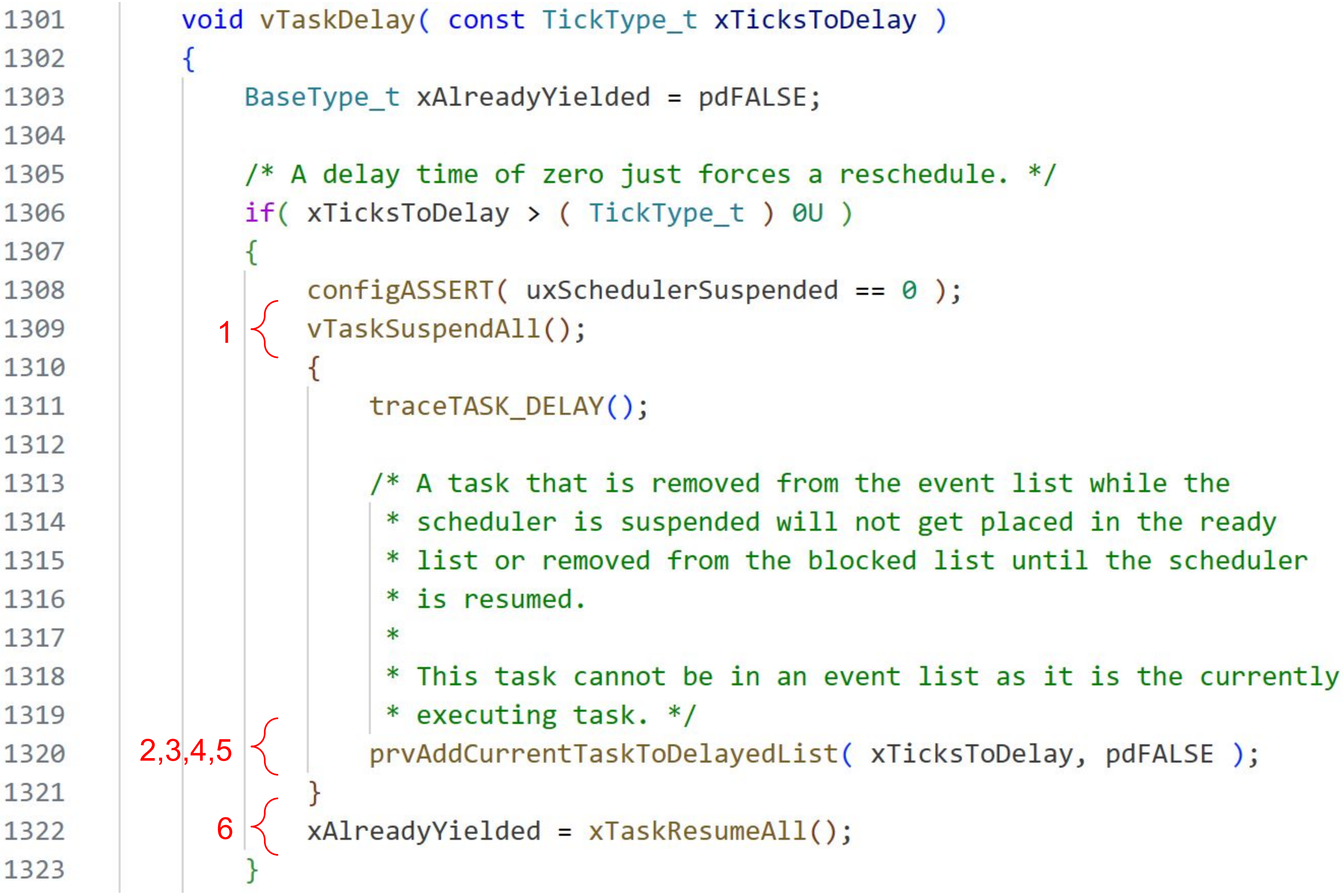


```
    if( pxNewTCB != NULL )
    {
        #if ( tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE != 0 ) /*lint !e9029 !e731 Macro has been consolidated for readability reasons. */
        {
            /* Tasks can be created statically or dynamically, so note this
             * task was created dynamically in case it is later deleted. */
            pxNewTCB->ucStaticallyAllocated = tskDYNAMICALLY_ALLOCATED_STACK_AND_TCB;
        }
        #endif /* tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE */

        prvInitialiseNewTask( pxTaskCode, pcName, ( uint32_t ) usStackDepth, pvParameters, uxPriority, pxCreatedTask, pxNewTCB, NULL );
        prvAddNewTaskToReadyList( pxNewTCB );
        xReturn = pdPASS;
    }
    else
    {
        xReturn = errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY;
    }

    return xReturn;
}
```

vTaskDelay

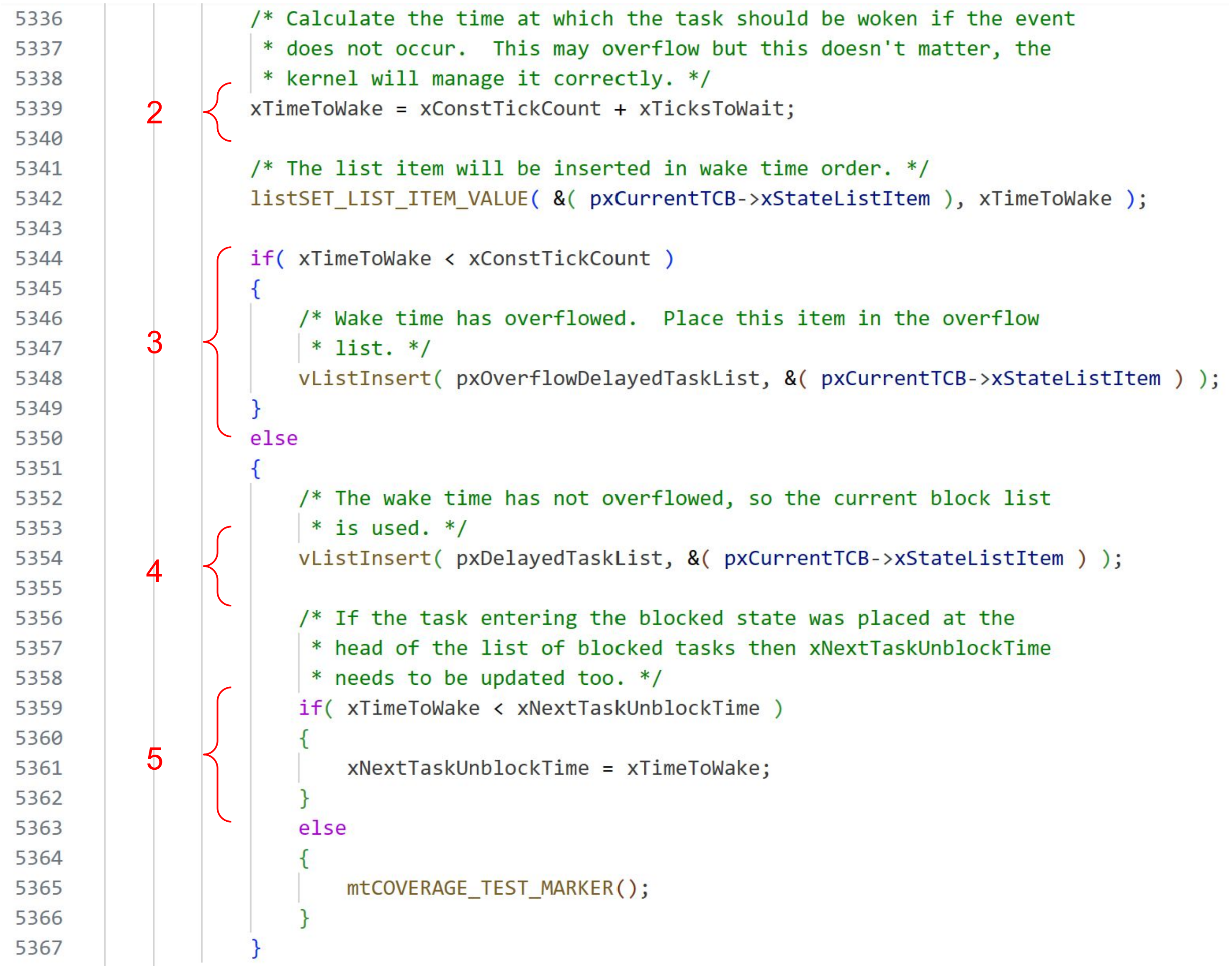
1. Suspend All task.(equivalently lock the scheduler)
2. Calculate the wake up time.
3. If the wake up time is overflow, add the task to **pxOverflowDelayedTaskList.**
4. Else add the task to **pxDelayedTaskList.**
5. Update **xNextTaskUnblockTime** if necessary.
6. Resume All the task.

vTaskDelay



```
    void vTaskDelay( const TickType_t xTicksToDelay )
    {
        BaseType_t xAlreadyYielded = pdFALSE;

        /* A delay time of zero just forces a reschedule. */
        if( xTicksToDelay > ( TickType_t ) 0U )
        {
            configASSERT( uxSchedulerSuspended == 0 );
            vTaskSuspendAll();
            {
                traceTASK_DELAY();

                /* A task that is removed from the event list while the
                 * scheduler is suspended will not get placed in the ready
                 * list or removed from the blocked list until the scheduler
                 * is resumed.
                 *
                 * This task cannot be in an event list as it is the currently
                 * executing task. */
                prvAddCurrentTaskToDelayedList( xTicksToDelay, pdFALSE );
            }
            xAlreadyYielded = xTaskResumeAll();
        }
```

vTaskDelay



```c
/* Calculate the time at which the task should be woken if the event
 * does not occur.  This may overflow but this doesn't matter, the
 * kernel will manage it correctly. */
xTimeToWake = xConstTickCount + xTicksToWait;

/* The list item will be inserted in wake time order. */
listSET_LIST_ITEM_VALUE( &( pxCurrentTCB->xStateListItem ), xTimeToWake );

if( xTimeToWake < xConstTickCount )
{
    /* Wake time has overflowed.  Place this item in the overflow
     * list. */
    vListInsert( pxOverflowDelayedTaskList, &( pxCurrentTCB->xStateListItem ) );
}
else
{
    /* The wake time has not overflowed, so the current block list
     * is used. */
    vListInsert( pxDelayedTaskList, &( pxCurrentTCB->xStateListItem ) );

    /* If the task entering the blocked state was placed at the
     * head of the list of blocked tasks then xNextTaskUnblockTime
     * needs to be updated too. */
    if( xTimeToWake < xNextTaskUnblockTime )
    {
        xNextTaskUnblockTime = xTimeToWake;
    }
    else
    {
        mtCOVERAGE_TEST_MARKER();
    }
}
```

FreeRTOS Lab

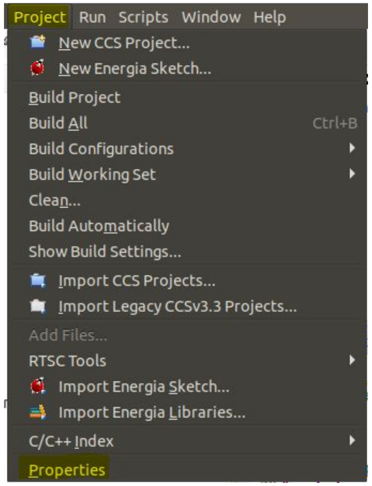
- This is Lab 4
- Redo your Lab1 (EDF) for FreeRTOS
- Contact the TA for borrowing a MSP430FR Launchpad
- Requirement: generate correct task scheduling events as you did in Lab1
- Check “FreeRTOS Lab” slice set for details

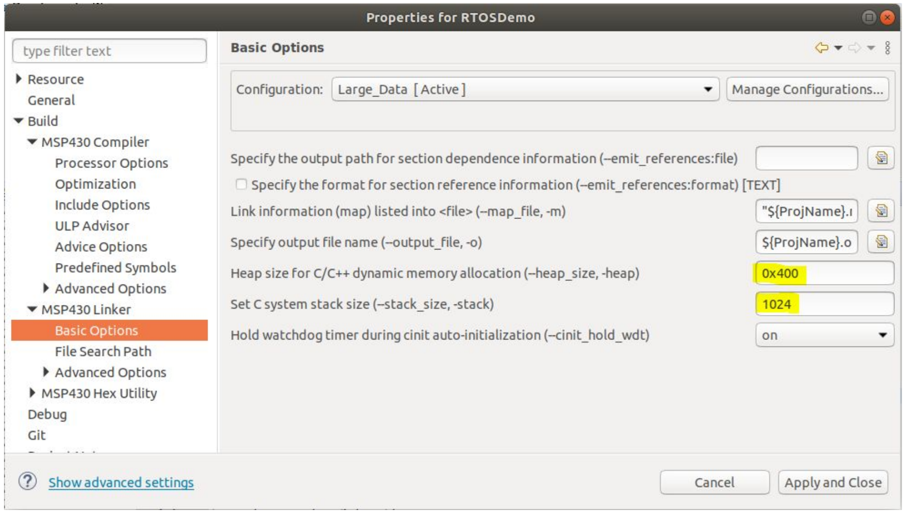
Debug Print

- You can also use breakpoints and memory watches for debugging
- For producing console output, there is a printf()
  - Okay to use it outside of ISRs
  - Don’t use in an ISR. Producing buggy results
- Reminder: to produce results for your lab
  - Log your events in a memory buffer and print out the results later in main.c
  - As you did with labs for uC/OS2

Setup for Using printf()

- Modify printf-stdarg.c
  - Comment out `#define putchar(c) c`
  - Add `#include <stdio.h>`
  - Comment out `sprintf()`、`snprintf()`、`write()`
- Set heap size and stack size at Project->Properties





Configuration: Large_Data [Active]

Specify the output path for section dependence information (--emit_references:file)

□ Specify the format for section reference information (--emit_references:format) [TEXT]

Link information (map) listed into <file> (--map_file, -m) "${ProjName}.ı

Specify output file name (--output_file, -o) ${ProjName}.o

Heap size for C/C++ dynamic memory allocation (--heap_size, -heap) 0x400

Set C system stack size (--stack_size, -stack) 1024

Hold watchdog timer during cinit auto-initialization (--cinit_hold_wdt) on

- Add `#include <stdio.h>` to your .c file