in port.c

```
portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode, void *pvParameters )
{
portSTACK_TYPE *pxOriginalTOS;

    pxOriginalTOS = pxTopOfStack;

    /* Setup the initial stack of the task.  The stack is set exactly as
    expected by the portRESTORE_CONTEXT() macro. */

    /* First on the stack is the return address - which in this case is the
    start of the task.  The offset is added to make the return address appear
    as it would within an IRQ ISR. */
    *pxTopOfStack = ( portSTACK_TYPE ) pxCode + portINSTRUCTION_SIZE;
    pxTopOfStack--;

    *pxTopOfStack = ( portSTACK_TYPE ) 0xaaaaaaaa;  /* R14 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) pxOriginalTOS; /* Stack used when task starts goes in R13. */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x12121212;  /* R12 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x11111111;  /* R11 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x10101010;  /* R10 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x09090909;  /* R9 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x08080808;  /* R8 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x07070707;  /* R7 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x06060606;  /* R6 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x05050505;  /* R5 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x04040404;  /* R4 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x03030303;  /* R3 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x02020202;  /* R2 */
    pxTopOfStack--;
    *pxTopOfStack = ( portSTACK_TYPE ) 0x01010101;  /* R1 */
    pxTopOfStack--;

    /* When the task starts is will expect to find the function parameter in
    R0. */
    *pxTopOfStack = ( portSTACK_TYPE ) pvParameters; /* R0 */
    pxTopOfStack--;

    /* The status register is set for system mode, with interrupts enabled.
*/
    *pxTopOfStack = ( portSTACK_TYPE ) portINITIAL_SPSR;

    #ifdef THUMB_INTERWORK
    {
        /* We want the task to start in thumb mode. */
        *pxTopOfStack |= portTHUMB_MODE_BIT;
    }
    #endif

    pxTopOfStack--;

    /* Interrupt flags cannot always be stored on the stack and will
    instead be stored in a variable, which is then saved as part of the
    tasks context. */
    *pxTopOfStack = portNO_CRITICAL_NESTING;

    return pxTopOfStack;
}
```

在stack上的context layout如下 (向下增长的stack)

address	contents
低地址	
+4	portNO_CRITICAL_NESTING (0)
+4	SPSR
+4	R0 (pvParameters)
+4	R1
+4	R2
+4	R3
+4	R4
+4	R5
+4	R6
+4	R7
+4	R8
+4	R9
+4	R10
+4	R11
+4	R12
+4	R13 (SP, original stack top)
+4	R14 (lr)
+4	function (pc)
(高地址)	

```
.macro portRESTORE_CONTEXT MACRO

    /* Set the lr to the task stack. */
    ldr     r1, =pxCurrentTCB           ①
    ldr     r0, [r1]                   ②
    ldr     lr, [r0]                   ③

    /* The critical nesting depth is the first item on the stack. */
    /* Load it into the ulCriticalNesting variable. */
    ldr     r0, =ulCriticalNesting        ④
    ldmfd       lr!, {r1}                 ⑤
    str     r1, [r0]                   ⑥

    /* Get the SPSR from the stack. */
    ldmfd       lr!, {r0}               ⑦
    msr     spsr_cxsf, r0              ⑧

    /* Restore all system mode registers for the task. */
    ldmfd       lr, {r0-r14}^           ⑨
    nop

    /* Restore the return address. */
    ldr     lr, [lr, #+60]           ⑩

    /* And return - correcting the offset in the LR to obtain the */
    /* correct address. */
    subs        pc, lr, #4

.endm
```

①

r1 = &pxCurrentTCB

在去掉无关成员后，TCB有如下成员

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

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

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

    #if ( configUSE_MUTEXES == 1 )
        UBaseType_t     uxBasePriority;
    #endif
} tskTCB;
```

②

r0 = pxCurrentTCB;

③

lr = *(pxCurrentTCB + 0)

即

lr = pxCurrentTCB->pxTopOfStack;

④

r0 = &ulCriticalNesting;

⑤

lr = pxCurrentTCB->pxTopOfStack;

r1 = [lr] = *(pxCurrentTCB->pxTopOfStack) = portNO_CRITICAL_NESTING (0)

同时

lr = pxCurrentTCB->pxTopOfStack ++

⑥

ulCriticalNesting = portNO_CRITICAL_NESTING (0)

⑦

r0 = SPSR

lr = pxCurrentTCB->pxTopOfStack ++

⑧

move R0 ==> spsr_cxsf

restore CPSR

⑨

restore r0 - r14 register

这时lr(r14)也被restore了!

⑩
???

```
.macro portSAVE_CONTEXT MACRO

    /* Push r0 as we are going to use the register. */
    stmdb       sp!, {r0}

    /* Set R0 to point to the task stack pointer. */
    stmdb       sp, {sp}^
    nop
    sub     sp, sp, #4
    ldmia       sp!, {r0}

    /* Push the return address onto the stack. */
    stmdb       r0!, {lr}

    /* Now we have saved lr we can use it instead of r0. */
    mov     lr, r0

    /* Pop r0 so we can save it onto the system mode stack. */
    ldmia       sp!, {r0}

    /* Push all the system mode registers onto the task stack. */
    stmdb       lr, {r0-lr}^
    nop
    sub     lr, lr, #60

    /* Push the spsr onto the task stack. */
    mrs     r0, spsr
    stmdb       lr!, {r0}

    ldr     r0, =ulCriticalNesting
    ldr     r0, [r0]
    stmdb       lr!, {r0}

    /* Store the new top of stack for the task. */
    ldr     r1, =pxCurrentTCB
    ldr     r0, [r1]
    str     lr, [r0]

.endm
```

```
SwiHandler_second_stage:
    add lr, lr, #4
    portSAVE_CONTEXT
    ldr     r0, =vTaskSwitchContext
    mov     lr, pc
    bx  r0
    portRESTORE_CONTEXT
```