Porting MicroC/OS-II

Santosh Sam Koshy
Member Technical Staff
C-DAC, Hyderabad

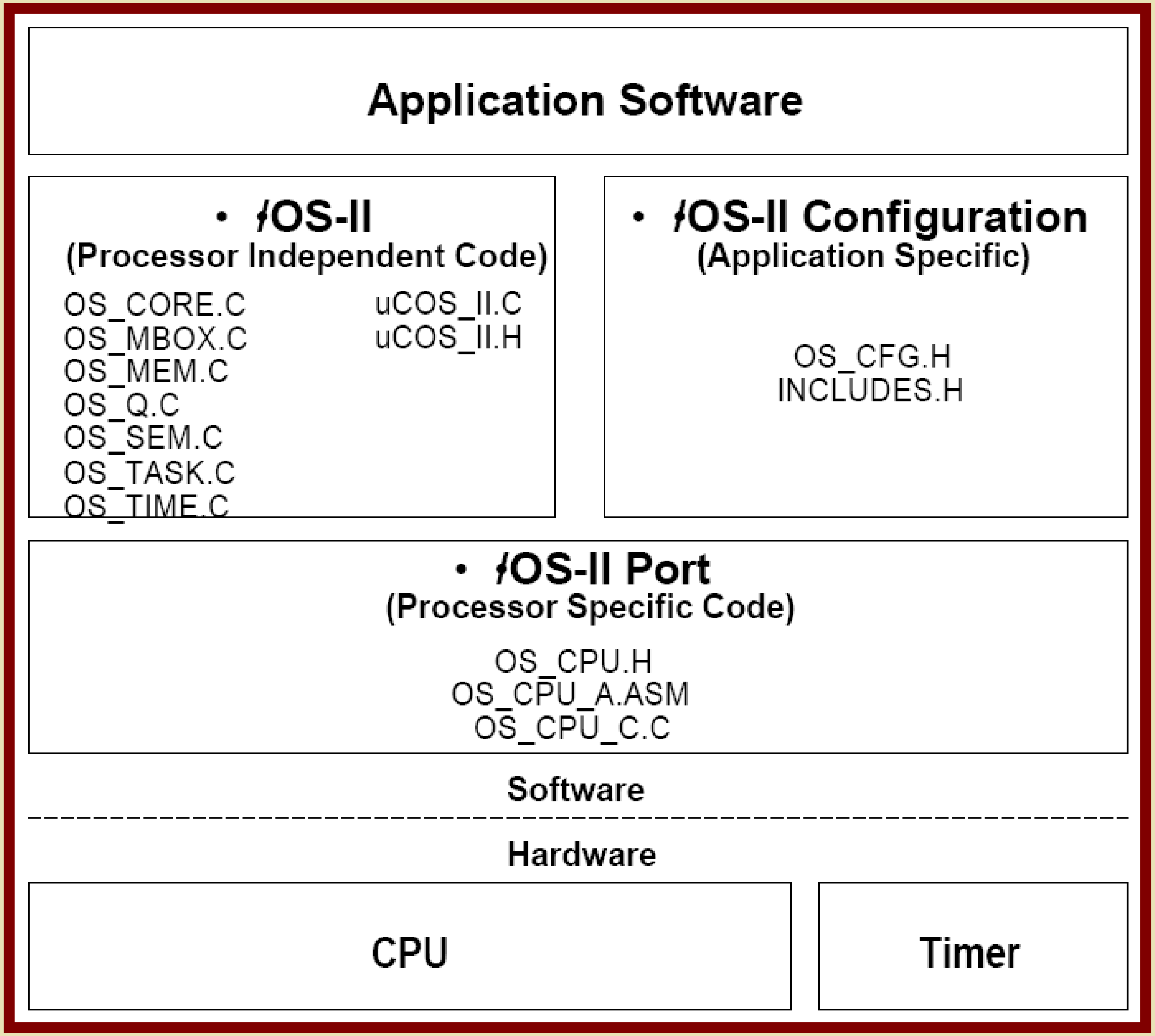
What do we mean by Porting?

Adapting a kernel to a microprocessor or a micro-controller is called a port.

Requirements to port uCOS-II

- *Re-entrant code*
- *Support for timely interrupts*
- *'C' control of interrupts*
- *Hardware stack*
- *Instructions to operate on the stack pointer.....or mechanism to access the CPU stack pointer*

Hardware/Software Architecture



Application Software

- /OS-II

(Processor Independent Code)

OS_CORE.C
OS_MBOX.C
OS_MEM.C
OS_Q.C
OS_SEM.C
OS_TASK.C
OS_TIME.C

uCOS_II.C
uCOS_II.H

- /OS-II Configuration

(Application Specific)

OS_CFG.H
INCLUDES.H

- /OS-II Port

(Processor Specific Code)

OS_CPU.H
OS_CPU_A.ASM
OS_CPU_C.C

Software

Hardware

CPU

Timer

Port Summary

Name	Type	File	C/ASM	Complexity
BOOLEAN	Data Type	OS_CPU.H	C	1
INT8U	Data Type	OS_CPU.H	C	1
INT8S	Data Type	OS_CPU.H	C	1
INT16U	Data Type	OS_CPU.H	C	1
INT16S	Data Type	OS_CPU.H	C	1
INT32U	Data Type	OS_CPU.H	C	1
INT32S	Data Type	OS_CPU.H	C	1
FP32	Data Type	OS_CPU.H	C	1
FP64	Data Type	OS_CPU.H	C	1
OS_STK	Data Type	OS_CPU.H	C	2
OS_CPU_SR	Data Type	OS_CPU.H	C	2
OS_CRITICAL_METHOD	#define	OS_CPU.H	C	3
OS_STK_GROWTH	#define	OS_CPU.H	C	1
OS_ENTER_CRITICAL()	Macro	OS_CPU.H	C	3
OS_EXIT_CRITICAL()	Macro	OS_CPU.H	C	3
OSStartHighRdy()	Function	OS_CPU_A.ASM	ASM	2
OSCtxSw()	Function	OS_CPU_A.ASM	ASM	3
OSTickISR()	Function	OS_CPU_A.ASM	ASM	3
OSTaskStkInit()	Function	OS_CPU_A.ASM	ASM	3
OSInitHookBegin()	Function	OS_CPU_C.C	C	3
OSInitHookEnd()	Function	OS_CPU_C.C	C	1
OSTaskCreateHook()	Function	OS_CPU_C.C	C	1
OSTaskDelHook()	Function	OS_CPU_C.C	C	1
OSTaskSwHook()	Function	OS_CPU_C.C	C	1
OSTaskStatHook()	Function	OS_CPU_C.C	C	1

Five Steps for Porting UCOS

- *Setting the value of* **1 #define constants** *(OS_CPU.H)*
- *Declaring* **10 data types** *(OS_CPU.H)*
- *Declaring* **3 #define macros** *(OS_CPU.H)*
- *Writing* **6** *simple functions in C (OS_CPU_C.C)*
- *Writing* **4** *assembly language functions (OS_CPU_A.ASM)*

Testing the Port

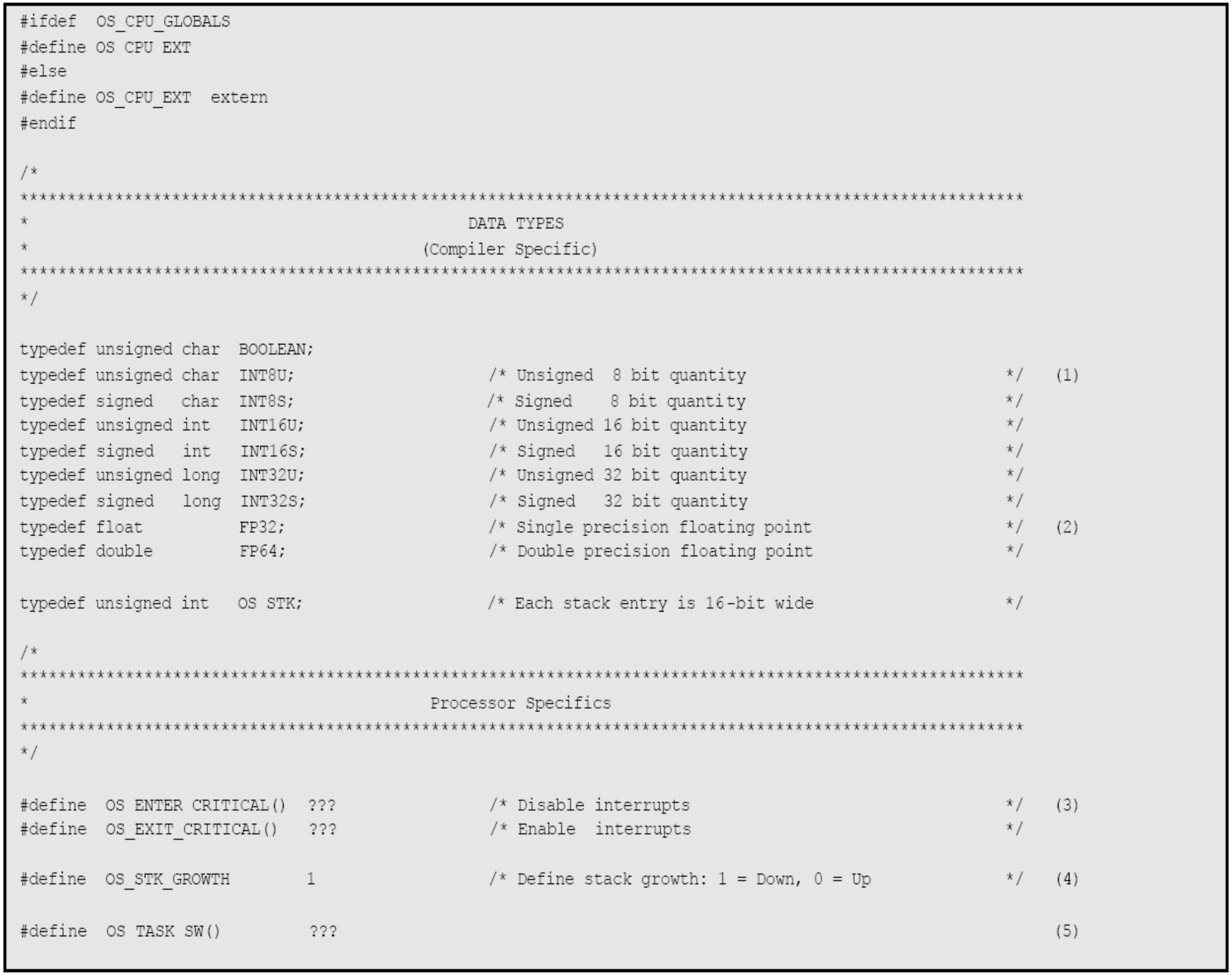
- *Test without Application Code*
  - *First, you don't want to complicate things anymore than they need to be*
  - *Second, if something doesn't work, you know that the problem lies in the port as opposed to your application*

INCLUDES.H

- *Master Header File*
- *Allows every .C file in your project to be written without concerns about which header file will actually be needed*
- *We can add our own header file*

OS_CPU.H

- *Processor & Implementation specific #define constants, macros and typedefs*



```
#ifdef  OS_CPU_GLOBALS
#define OS CPU EXT
#else
#define OS_CPU_EXT  extern
#endif

/*
*********************************************************************************************************
*                                              DATA TYPES
*                                         (Compiler Specific)
*********************************************************************************************************
*/

typedef unsigned char  BOOLEAN;
typedef unsigned char  INT8U;                    /* Unsigned  8 bit quantity                           */   (1)
typedef signed   char  INT8S;                    /* Signed    8 bit quantity                           */
typedef unsigned int   INT16U;                   /* Unsigned 16 bit quantity                           */
typedef signed   int   INT16S;                   /* Signed   16 bit quantity                           */
typedef unsigned long  INT32U;                   /* Unsigned 32 bit quantity                           */
typedef signed   long  INT32S;                   /* Signed   32 bit quantity                           */
typedef float          FP32;                     /* Single precision floating point                    */   (2)
typedef double         FP64;                     /* Double precision floating point                    */

typedef unsigned int   OS STK;                   /* Each stack entry is 16-bit wide                    */

/*
*********************************************************************************************************
*                                           Processor Specifics
*********************************************************************************************************
*/

#define  OS ENTER CRITICAL()  ???                /* Disable interrupts                                 */   (3)
#define  OS_EXIT_CRITICAL()   ???                /* Enable  interrupts                                 */

#define  OS_STK_GROWTH        1                  /* Define stack growth: 1 = Down, 0 = Up              */   (4)

#define  OS TASK SW()         ???                                                                           (5)
```

Data Types

```
typedef     unsigned char      BOOLEAN;
typedef     unsigned char      INT8U;
typedef     signed   char      INT8S;
typedef     unsigned int       INT16U;
typedef     signed   int       INT16S;
typedef     unsigned long      INT32U;
typedef     signed   long      INT32S;
typedef     float              FP32;
typedef     unsigned char      OS_STK;
         /* Each stack entry is 8-bit wide */
```

OS_ENTER_CRITICAL() & OS_EXIT_CRITICAL()

- *Method I*
  - *#define OS_ENTER_CRITICAL asm(“cli”)*
  - *#define OS_ENTER_CRITICAL asm(“sei”)*
- *Method II*
  - *#define OS_ENTER_CRITICAL*

    ```
    asm(“push SREG ;\
              cli”)
    ```
  - *#define OS_ENTER_CRITICAL asm(“pop SREG”)*

How to do it in uCOS for ARM…..

- Define 2 special functions to cater to the disabling and enabling of interrupts
- #define OS_CRITICAL METHOD 3
- **#define OS_ENTER_CRITICAL { cpu_sr = OS_CPU_SR_Save ();}**
- **#define OS_EXIT_CRITICAL {OS_CPU_SR_Restore (cpu_sr);}**

- Note:
  - On calling a function the arguments to the function are stored in registers R0, R1, R2 and R3
  - On returning from a function, the return type is collected from R0.
  - All these changes have to be caused in the os_cpu.h file

Code Snippet

```
OS_CPU_SR_Save:
        MRS    R0, CPSR
        ORR    R1, R0, #NO_INT
        MSR    CPSR_c, R1
        MRS   R1, CPSR
        AND    R1, R1, #NO_INT
        CMP    R1, #NO_INT
        BNE    OS_CPU_SR_Save
        BX          LR
OS_CPU_SR_Restore:
        MSR   CPSR_c, R0
        BX     LR
```

OS_STK_GROWTH

- *Some processor will have a Stack Growth from High to Low or from Low to High*
- *In case of AVR/ARM (High to Low)*
  - *#define OS_STK_GROWTH 1*

OS_CPU_C.C

- *OSTaskStkInit()*
- *OSTaskCreateHook()*
- *OSTaskDelHook()*
- *OSTaskSwHook()*
- *OSTaskStatHook()*
- *OSTimeTickHook()*

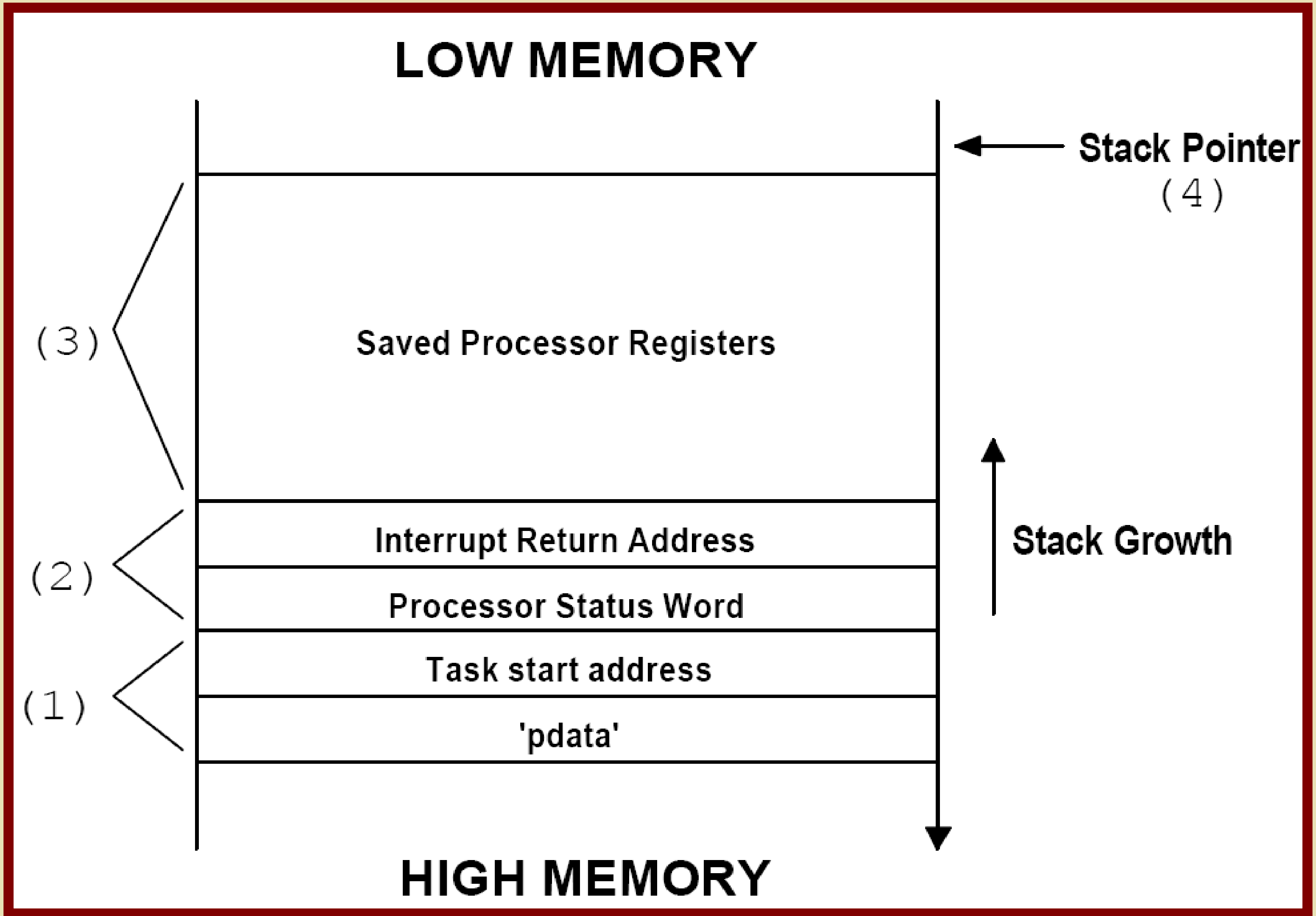
OSTaskStkInit()

- *Called whenever a new task is created*
- *Duties*
  - *Initialize the stack for the newly created task*
  - *Set the environment such that the task may be brought into execution as though an interrupt has occurred*
  - *Return the initialized top-of-stack*
- *Note*
  - *Critical that you understand the context switch flow for your processor*

OSTaskStkInit()

```
OS_STK *OSTaskStkInit (void (*task) (void *pd),
                       void *pdata, OS_STK *ptos,
                       INT16U opt);
{
   Simulate call to function with an argument (i.e.
      Pdata);
   Simulate ISR vector;
   Setup stack frame to contain desired initial values
      of all registers
   Return new top-of-stack pointer to caller
}
```

OSTaskStkInit()



Code Snippet

```
OS_STK  *OSTaskStkInit  (void  (*task)(void  *pd),
  void *p_arg, OS_STK *ptos, INT16U opt)
{
    OS_STK *stk;
    opt       = opt;
    stk       = ptos;
    *(stk)    = (OS_STK)task;
    *(--stk) = (INT32U)0x14141414L;
    *(--stk) = (INT32U)0x12121212L;
    *(--stk) = (INT32U)0x02020202L;
    *(--stk) = (INT32U)0x01010101L;
    *(--stk) = (INT32U)p_arg;
    *(--stk) = (INT32U)ARM_SVC_MODE;

    return (stk);
}
```

OSTaskCreateHook()

- *Called whenever OSTaskCreate() or OSTaskCreateExt() are used*
- *Called after setting up the Internal Data Structure but before Scheduling*
- *Interrupt are disabled before calling*
- *Hence code in this function should be as small as possible(Affects Interrupt Latency directly)*

OSTaskDelHook()

- *Called whenever a task is deleted*
- *It is called before MicorC/OS unlinks the internal data structure from linked list*

OSTaskSwHook()

- *Called whenever a task switch occurs*
- *This happens whether task switch is from OSCtxSw() or OSIntCtxSw()*
- *It can directly access OSTCBCur & OSTCBHighRdy (since global)*
- *Interrupts are disabled*

OSTaskStatHook()

- *Is called by OSTaskStat() every one second*
- *We could keep track and display the execution time of each task, the percentage of the CPU that is used by each task, how often each task executes and more*

OSTimeTickHook()

- *Is called by OsTimeTick() at every System Tick*
- *This function is called before MicroC/OS processes the tick*
- *Hence helps in giving application the first claim on tick*

OS_CPU_A.ASM

- *OSStartHighRdy()*
- *OSCtxSw()*
- *OSIntCtxSw()*
- *OSTickISR()*

OSStartHighRdy()

- Called into execution for the very first task that is started after the Multi-tasking kernel is started
- Duties
  - Load the CPU stack pointer with the address of the stack pointer for the task to be scheduled
  - Restore the registers according to the defined protocol
  - execute a return from interrupt
- Note
  - Careful about the popping order……

OSStartHighRdy()

- *Call OSTaskSwHook()*
- *Get Stack Pointer of the task to resume*
  - ***Stack Pointer = OSTCBHighRdy->OSTCBStkPtr;***
- *OSRunning = TRUE ;*
- *Restore all processor registers from the new Task's Stack*
- *Execute Return from Interrupt*

OS_TASK_SW

- *Represents the flow of events during a task level context switch……..Therefore, instrument this macro to trigger a software interrupt on the processor*

- *If our processor does not have an Software Interrupt we can directly make a call to OSCtxSw() function*

- *In AVR/ARM*
  - *#define OS_TASK_SW   OSCtxSw()*

OSCtxSw()

- Handles the task level context switch in Micro C OS. Runs in response to the function OS_TASK_SW()
- Duties
  - Save the present context of the task being preempted
  - Manage OS pointers to represent newly scheduled task
  - Retrieve the context of the task scheduled
  - Return from interrupt
- Note
  - Both context saving and retrieval are done here

OSCtxSw()

- *Save processor registers*
- *Save the current task's stack pointer into the current task's OS_TCB:*
  - ***OSTCBCur->OSTCBStkPtr = Stack pointer***
- *Call user definable OSTaskSwHook()*
- ***OSTCBCur = OSTCBHighRdy***
- ***OSPrioCur = OSPrioHighRdy***
- *Get the stack pointer of the task to resume*
  - ***Stack pointer = OSTCBHighRdy->OSTCBStkPtr***
- *Restore all processor registers from the new task's stack*
- *Execute a return from interrupt instruction*

OSIntCtxSw()

- *Handles the context switching property from interrupt context*
- *Duties*
  - *Check the condition if a new task has to be scheduled*
  - *Restore the context of the task to be scheduled*
  - *Return from Interrupt*
- *Note*
  - *Remember to check the schedule flag*
  - *Only retrieving the context is carried out here. Saving context has already been performed*

OSIntCtxSw()

- *Check whether a context switch is required.*
- *Save the current task's stack pointer into the current task's OS_TCB*
    - *OSTCBCur->OSTCBStkPtr = Stack pointer*
- *Call user definable OSTaskSwHook()*
- *OSTCBCur = OSTCBHighRdy*
- *OSPrioCur = OSPrioHighRdy*
- *Get the stack pointer of the task to resume*
    - *Stack pointer = OSTCBHighRdy->OSTCBStkPtr*
- *Restore all processor registers from the new task's stack*
- *Execute a return from interrupt instruction*

OSTickISR()

- *Save processor registers;*
- *Call OSIntEnter() or increment OSIntNesting;*
- *If OSIntNesting is 1, save the current SP to the task's TCB*
- *Call OSTimeTick();*
- *Call OSIntExit();*
- *Restore processor registers;*
- *Execute a return from interrupt instruction;*

Thank You