ECE254 Lab3 Tutorial

Introduction to Keil LPC1768 Hardware and Programmers Model

Irene Huang

(last updated: 2012/11/04)

Lab3 Part A Requirements (1)

- A function to obtain the task information

```
OS_RESULT os_tsk_get(OS_TID task_id, RL_TASK_INFO *buffer)
```

- Task information data structure

```
typedef struct rl_task_info {
  U8   state;           /* Task state                              */
  U8   prio;            /* Execution priority                      */
  U8   task_id;         /* Task ID value for optimized TCB access  */
  U8   stack_usage;     /* Stack usage percent value. eg.=58 if 58%*/
  void (*ptask)();      /* Task entry address                      */
} RL_TASK_INFO;
```

- Return value
  - OS_R_OK
  - OS_R_NOK

Lab3 Part A Requirements (2)

- Task state symbols

```
#define INACTIVE        0
#define READY           1
#define RUNNING         2
#define WAIT_DLY        3
#define WAIT_ITV        4
#define WAIT_OR         5
#define WAIT_AND        6
#define WAIT_SEM        7
#define WAIT_MBX        8
#define WAIT_MUT        9
#define WAIT_MEM        10
```

- Assumption
  - No user defined stack in any tasks in the system.

Lab3 Requirements Part B (1)

- A blocking fixed-size memory allocator

```
void *os_mem_alloc(unsiged char flag)
```

  - The flag takes two values

`MEM_NOWAIT` (sample implementation available)

Return value: a pointer to a fixed-size memory block if there is a free memory block
NULL if there is no free memory block.

`MEM_WAIT` (you need to implement this one)

Return value: a pointer to a fixed-size memory block if there is a free memory block
blocks the calling task until there is a free memory available.

- A function to release the memory block

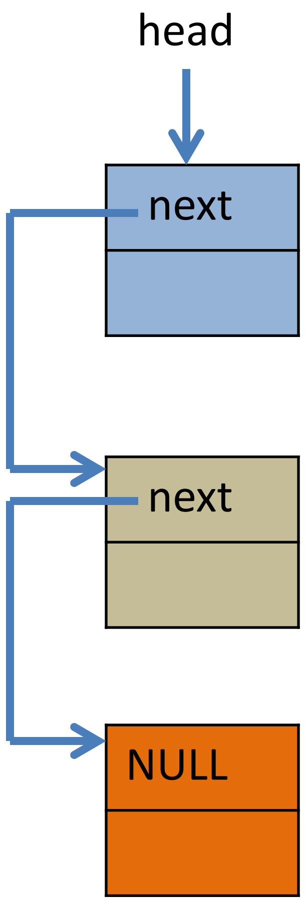
```
OS_RESULT os_mem_free(void *ptr)
```

  - You will need to modify the existing implementation to handle the case when a process is blocked waiting for memory.
  - Return
    - `OS_R_OK`
    - `OS_R_NOK`

Lab3 Requirements Part B (2)

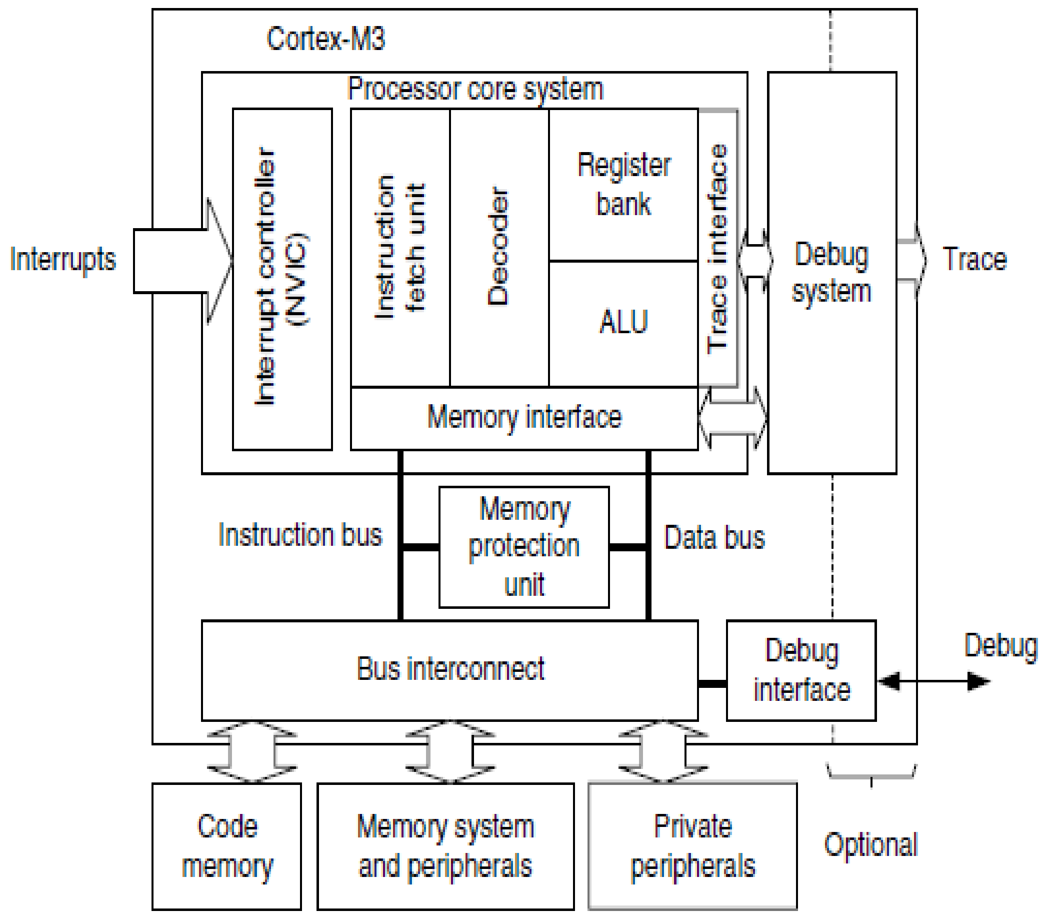
- How do you initialize a memory region?

Description	Address	Value	Code
Meta Data	0x100013B0:	0x100013BC	(P_BM) p->free
	0x100013B4:	0x100013D4	(P_BM) p->end
	0x100013B8:	0x00000008	(P_BM) p->blk_size
block 1	0x100013BC:	0x100013C4	
	0x100013C0:		
block 2	0x100013C4:	0x100013CC	
	0x100013C8:		
block 3	0x100013CC:	0x00000000	
	0x100013D0:		



A memory pool with three blocks. Each block is eight byte long.

Cortex-M3 Overview



(Image Courtesy of [1])

- 32-bit microprocessor
  - 32-bit data path
  - 32-bit register bank
  - 32-bit memory interface
- Harvard Architecture
  - Separate data and memory bus
  - instruction and data buses share the same memory space (a unified memory system)

Cortex-M3 Registers

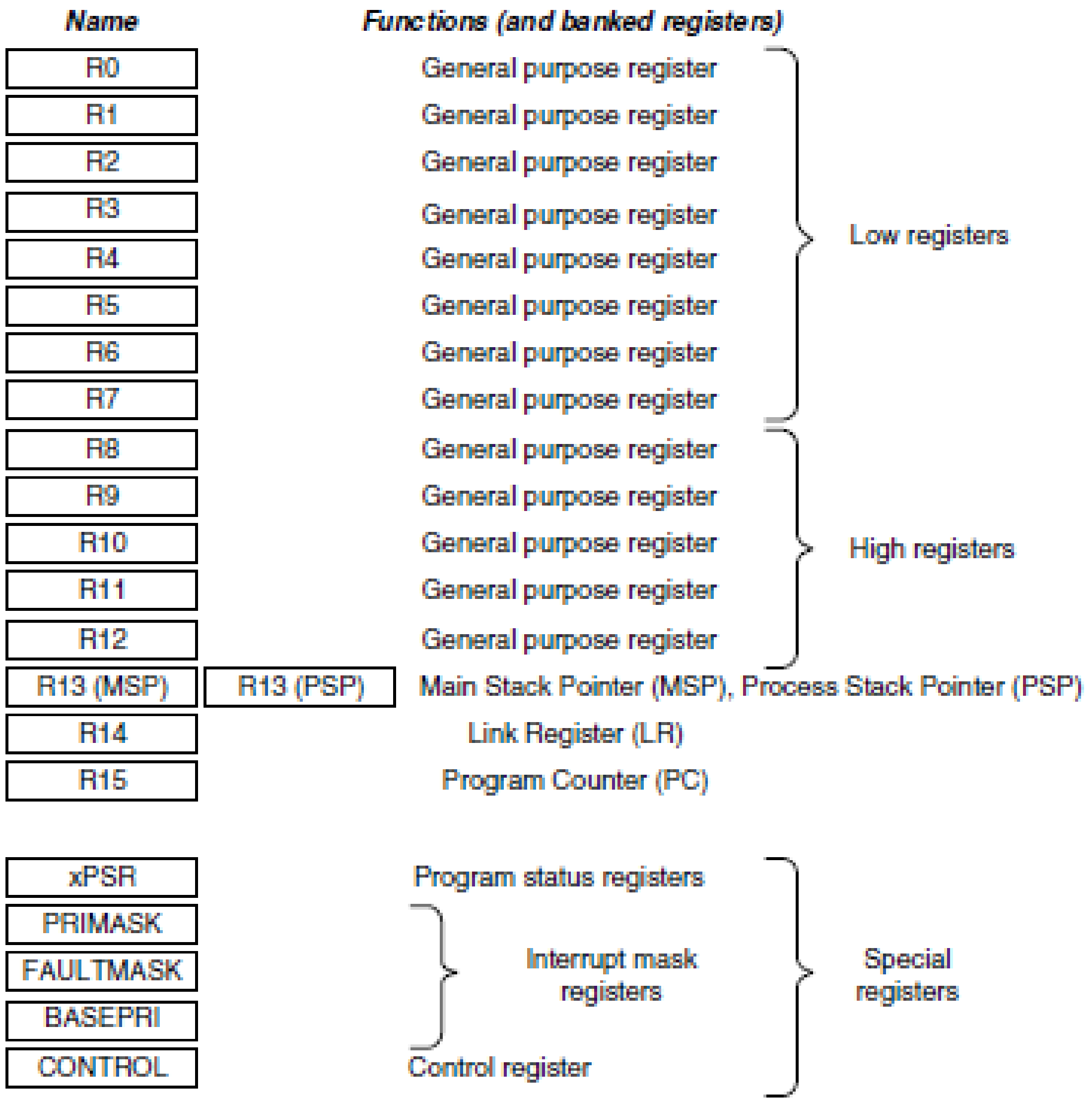
- General Purpose Registers (R0-R15)
  - Low registers (R0-R7)
    - 16-bit Thumb instructions and 32-bit Thumb-2 instructions
  - High registers (R8-R12)
    - All Thumb-2 instructions
  - Stack Pointer (R13)
    - **MSP:** Privileged, default after reset, os kernel, exception handler
    - **PSP:** Uer-level (i.e. unprivileged) base-level application
  - Link Register (R14)
  - Program Counter (R15)
- Special Registers
  - Program Status registers (PSRs)
  - Interrupt Mask registers (PRIMASK, FAULTMASK, and BASEPRI)
  - Control register (CONTROL)

Cortex-M3 Registers

32-bit microprocessor
32-bit data path
32-bit register bank
32-bit memory interface
Harvard Architecture
Separate data and memory bus

Low registers: R0-R7
16-bit Thumb instructions
32-bit Thumb-2 instructions
High registers: R9-R12
All Thumb-2 instructions

MSP: default after reset
os kernel, exception handler
Privileged
PSP: base-level application
unprivileged, user-level

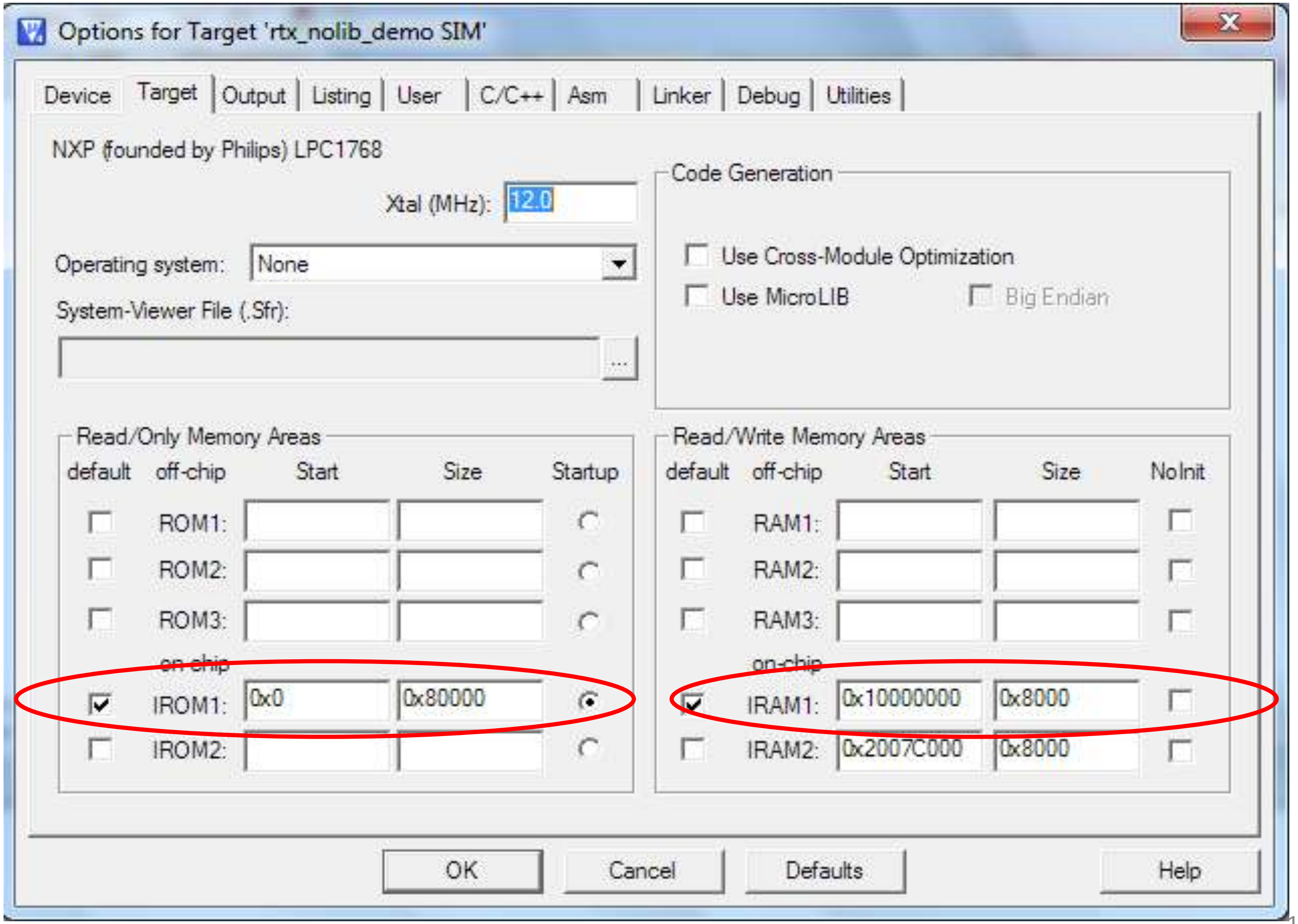


(Image Courtesy of [1])

LPC1768 Memory Map

`0x2008 4000`		
`0x2007 C000`	32 KB	AHB SRAM (2 blocks of 16 KB)
`0x1FFF 2000`		Reserved
`0x1FFF 0000`	8 KB	Boot ROM
`0x1000 8000`		Reserved
`0x1000 0000`	32 KB	**Local SRAM**
`0x0008 0000`		Reserved
`0x0000 0000`	512 KB	On-chip flash

Memory Configuration



Options for Target 'rtx_nolib_demo SIM'

NXP (founded by Philips) LPC1768

Xtal (MHz): 12.0

Operating system: None

System-Viewer File (.Sfr):

Code Generation

- [ ] Use Cross-Module Optimization
- [ ] Use MicroLIB
- [ ] Big Endian

Read/Only Memory Areas

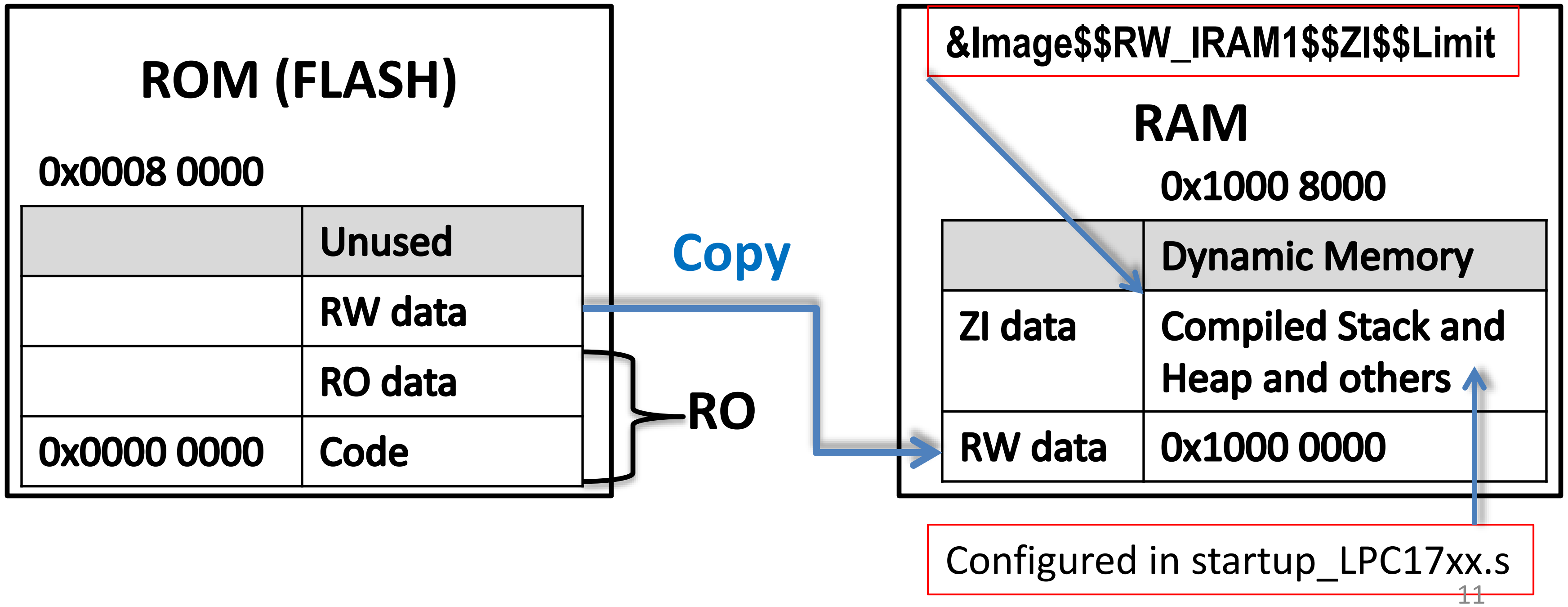
default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x0	0x80000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x10000000	0x8000	☐
☐	IRAM2:	0x2007C000	0x8000	☐

Image memory layout

- A simple image consists of:
  - read-only (RO) section (Code + RO-data)
  - a read-write (RW) section (RW-data)
  - a zero-initialized (ZI) section (ZI-data)



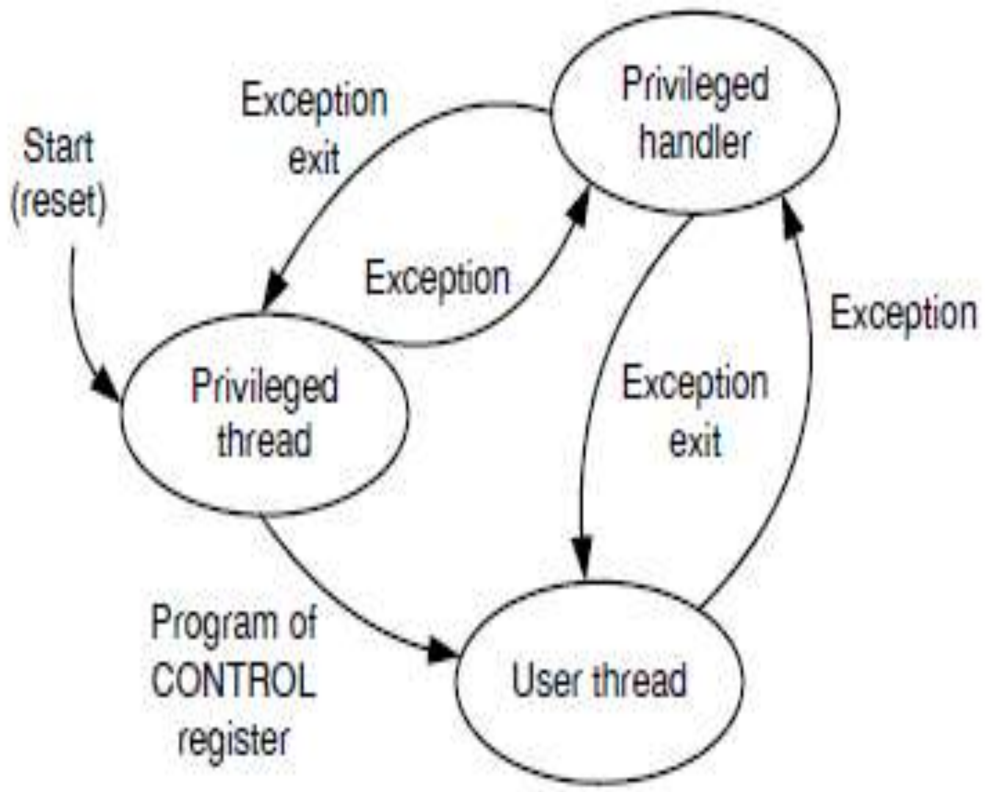
End Address of the Image

- Linker defined symbol Image$$RW_IRAM1$$ZI$$Limit

```
extern unsigned int Image$$RW_IRAM1$$ZI$$Limit;

unsigned int free_mem =
    (unsigned int) &Image$$RW_IRAM1$$ZI$$Limit;
```

Operation Modes

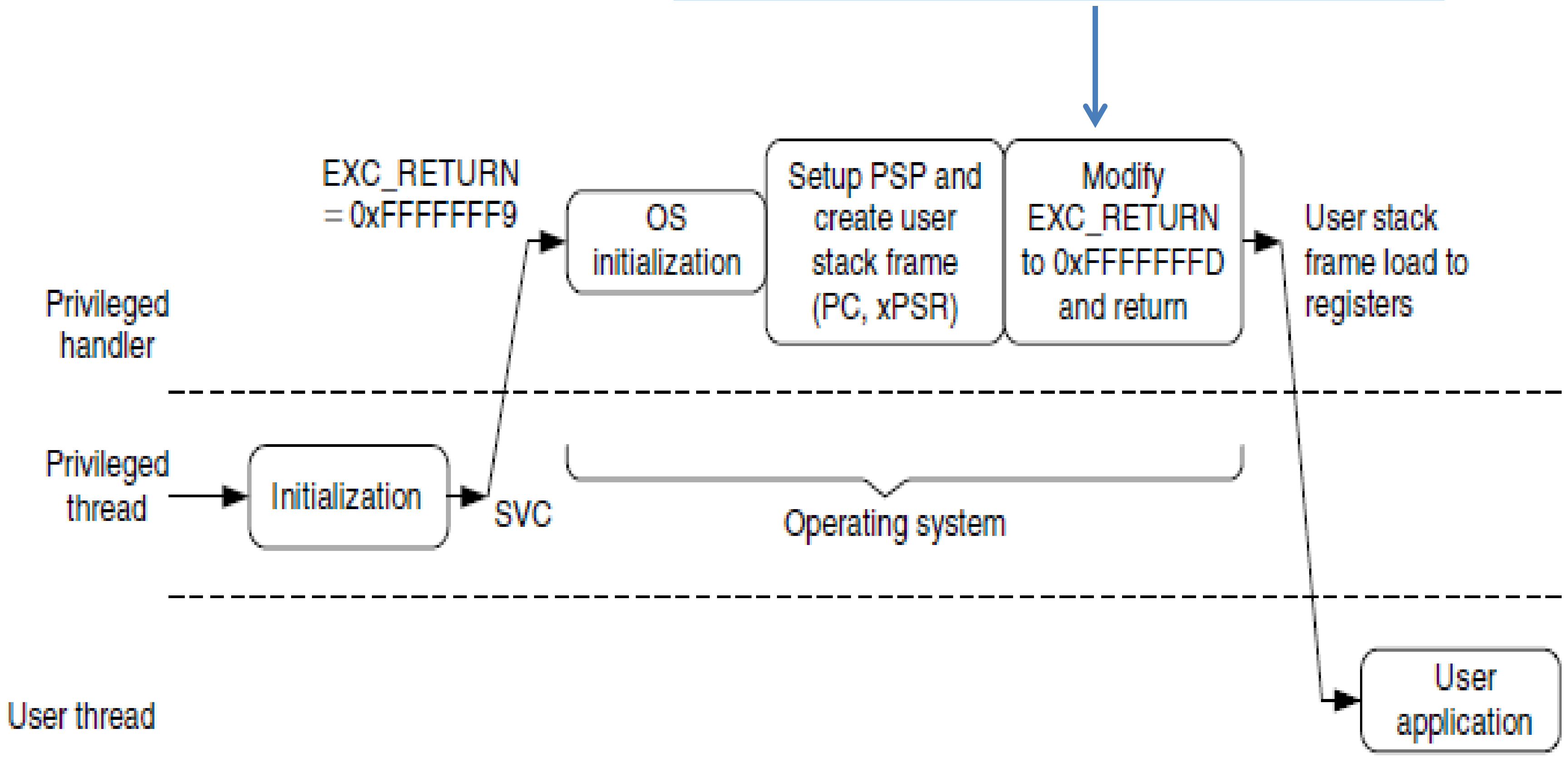
- Two modes
  - Thread mode
  - Handler mode
- Two privilege levels
  - Privileged level
  - User level



(Image Courtesy of [1])

OS Initialization Mode Switch

When MSP is used for user application, then EXC_RETURN=0xFFFFFFF9



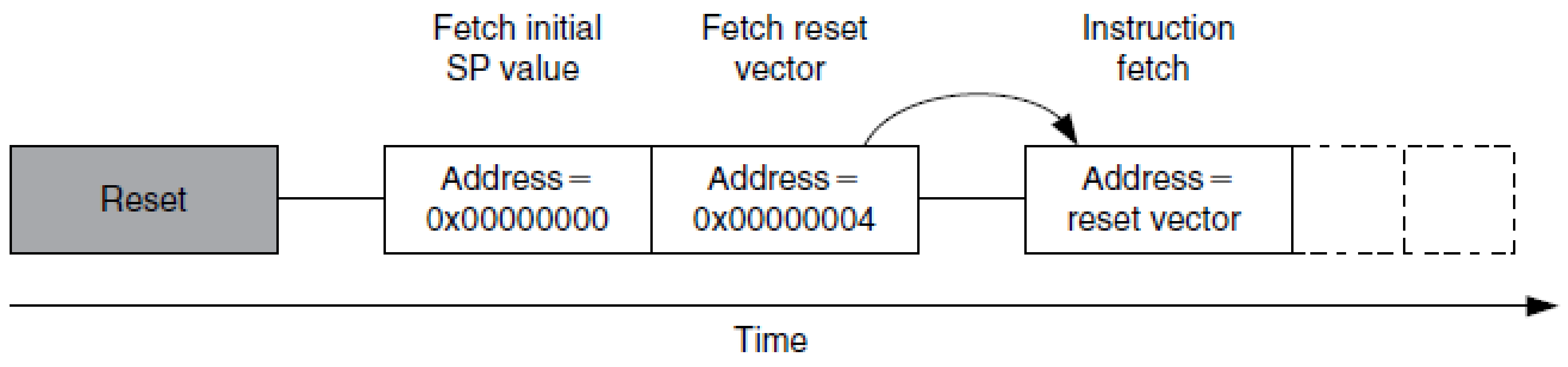
(Image Courtesy of [1])

Exceptions (1)

- NVIC (Nested Vectored Interrupt Controller)
  - System Exceptions
    - Exception Numbers 1 -15
    - SVC call exception number is 11
    - SysTick exception number is 15
  - External Interrupts
    - Exception Numbers 16-50
    - Timer0-3 IRQ numbers are 17-20
    - UART0-3 IRQ numbers are 21-24
- Vector table is at 0x0 after reset.
- 32 programmable priorities
- Each vector table entry contains the exception handler's address (i.e. entry point)

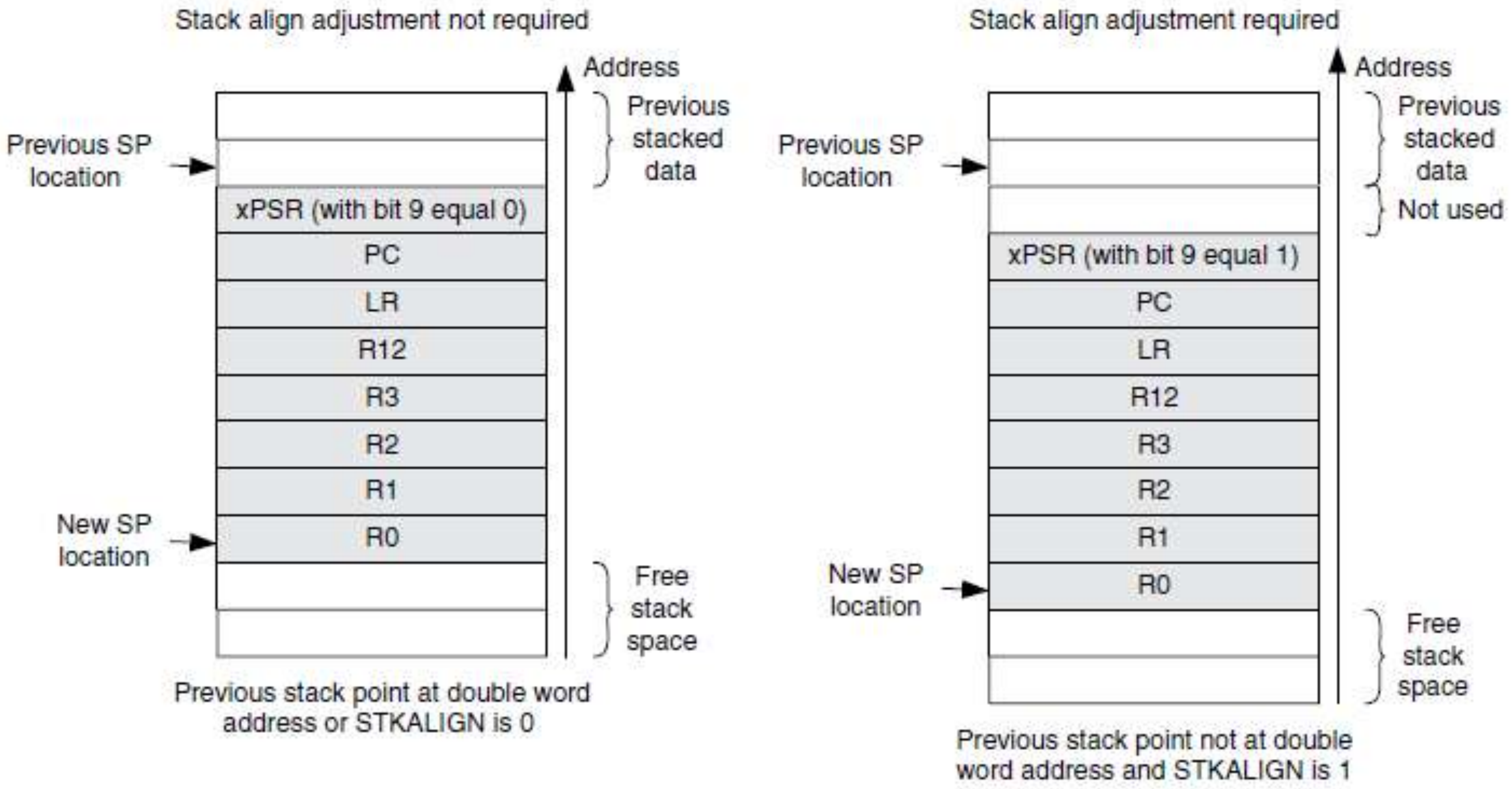
Exceptions (2)

Address	Exception Number	Value (Word Size)
0x0000 0000	-	MSP initial value
0x0000 0004	1	Reset vector (program counter initial value)
0x0000 0008	2	NMI handler starting address
0x0000 000C	3	Hard fault handler starting address
…	…	Other handler starting address



(Image Courtesy of [1])

Exception Stack Frame

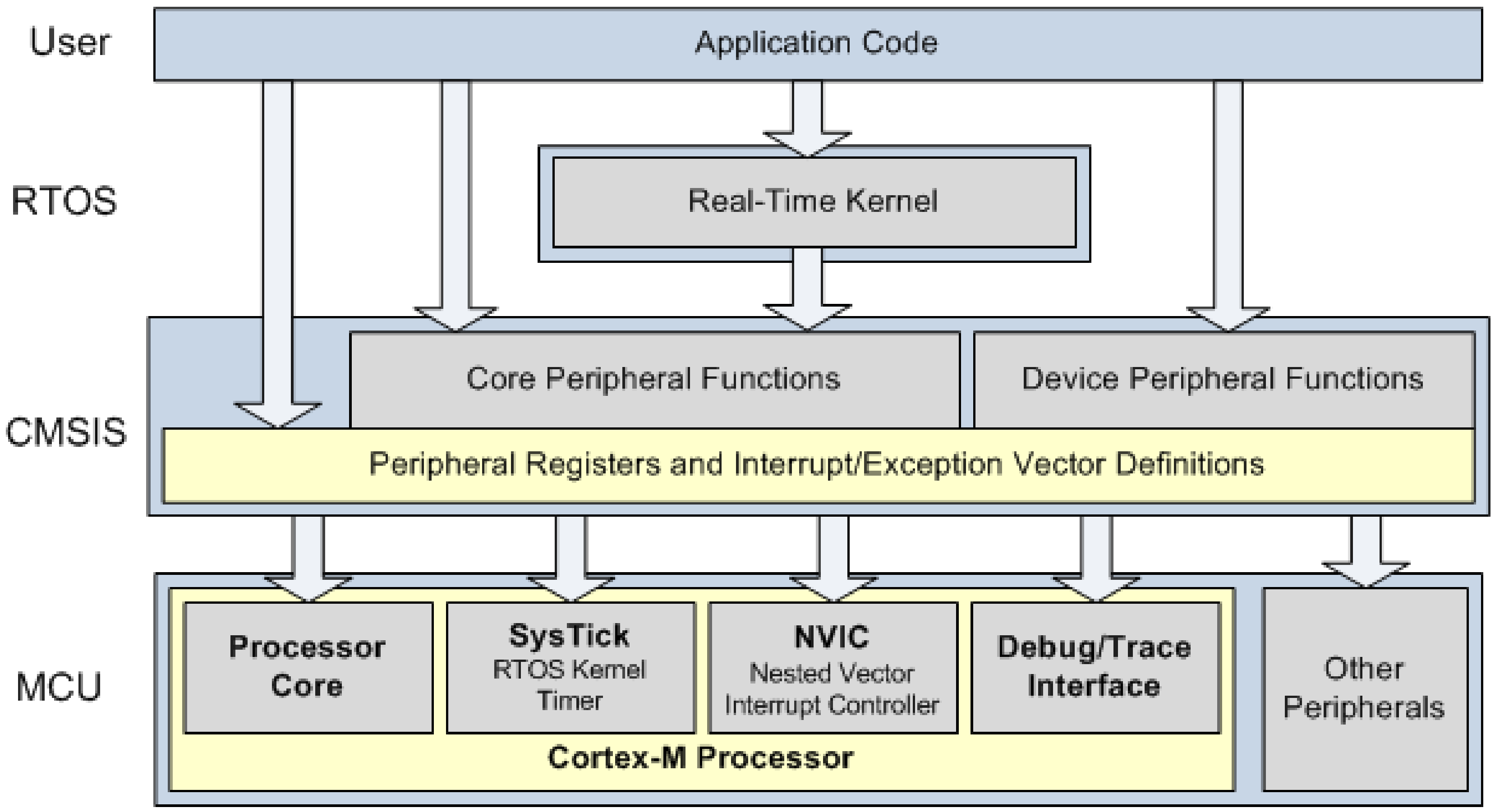

Stack align adjustment not required
Address
Previous stacked data
Previous SP location
xPSR (with bit 9 equal 0)
PC
LR
R12
R3
R2
R1
R0
New SP location
Free stack space
Previous stack point at double word address or STKALIGN is 0
Stack align adjustment required
Address
Previous stacked data
Not used
Previous SP location
xPSR (with bit 9 equal 1)
PC
LR
R12
R3
R2
R1
R0
New SP location
Free stack space
Previous stack point not at double word address and STKALIGN is 1

(Image Courtesy of [1])

AAPCS (ARM Architecture Procedure Call Standard)

- R0-R3 , R12
  - Input parameters Px of a function. R0=P1, R1=P2, R2=P3 and R3=P4
  - **R0** is used for **return value** of a function
- R12, SP, LR and PC
  - R12 is the Intra-Procedure-call scratch register.
- R4-R11
  - Must be preserved by the called function. C compiler generates push and pop assembly instructions to save and restore them automatically.

CMSIS Structure


User
Application Code
RTOS
Real-Time Kernel
CMSIS
Core Peripheral Functions
Device Peripheral Functions
Peripheral Registers and Interrupt/Exception Vector Definitions
MCU
Processor Core
SysTick RTOS Kernel Timer
NVIC Nested Vector Interrupt Controller
Debug/Trace Interface
Cortex-M Processor
Other Peripherals

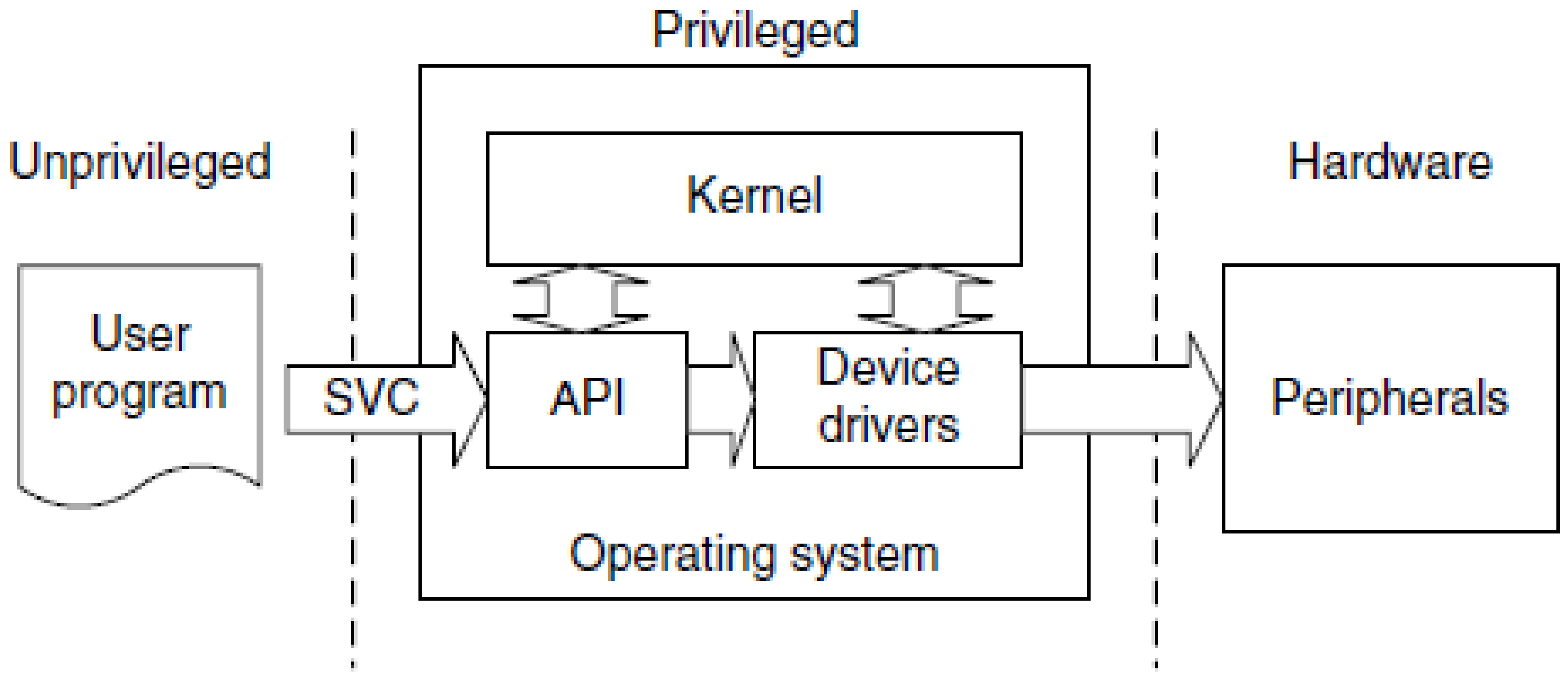
(Image Courtesy of MDK-ARM Primer V4.60)

Exception Handler Programming

- Hardware Abstraction Layer file: `HAL_CM3.c`

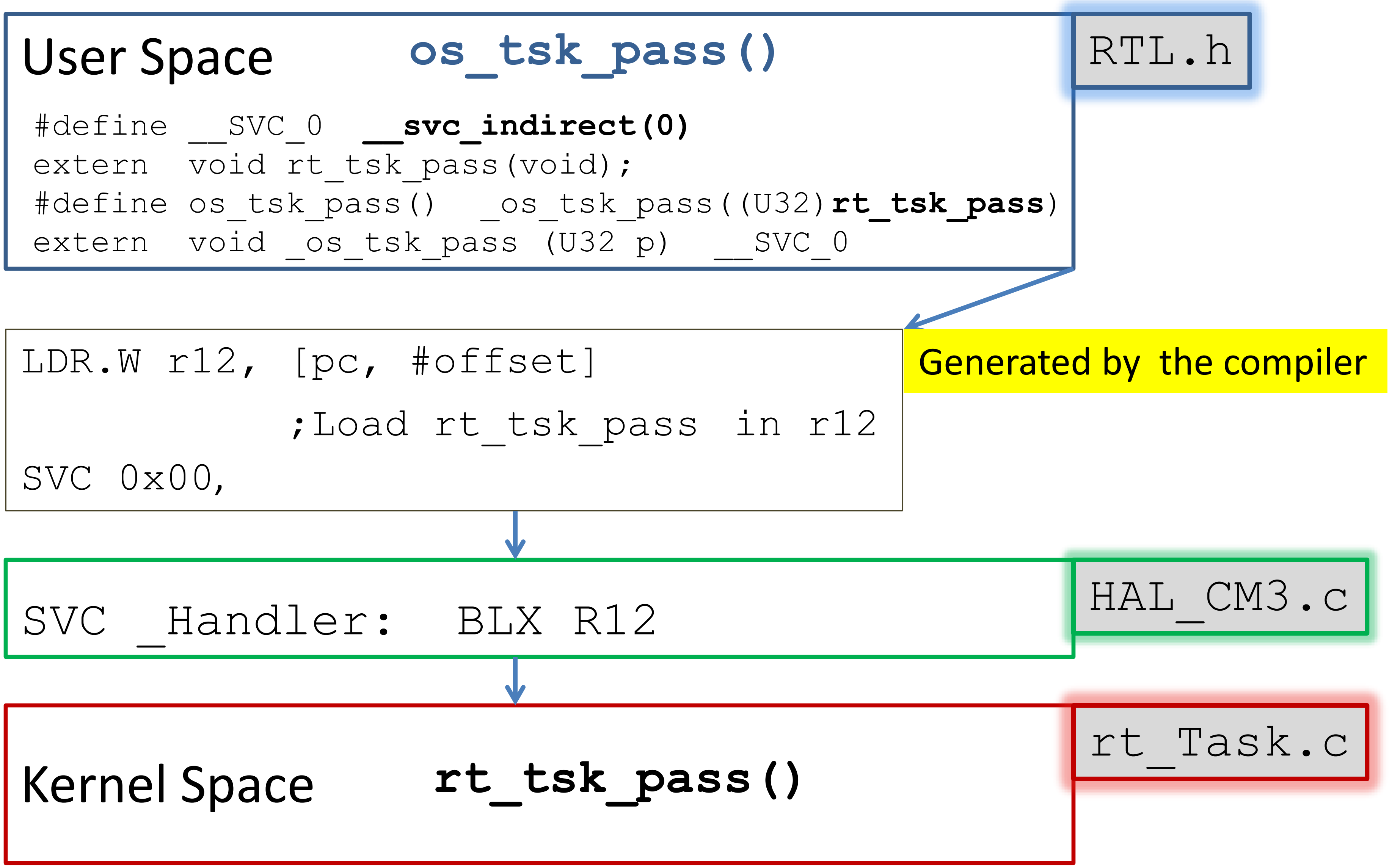
```
__asm void SVC_Handler (void) {
    PRESERVE8                    ;8 byte alignment of the stack

    IMPORT  SVC_Count            ; an external ASM symbol
    IMPORT  SVC_Table            ; an external ASM symbol
    IMPORT  rt_stk_check         ; an external C symbol

    MRS     R0,PSP               ; Read PSP
    LDR     R1,[R0,#24]          ; Read Saved PC from Stack
    LDRB    R1,[R1,#-2]          ; Load SVC Number
    CBNZ    R1,SVC_User          ; if SVC# != zero, goto SVC_User

    LDM     R0,{R0-R3,R12};      ; Read R0-R3,R12 from stack
    BLX     R12                  ; Call SVC Function
    ; omit the rest of the code below
}
```

SVC as a Gateway for OS Functions



(Image Courtesy of [1])

System calls through SVC in C



User Space `os_tsk_pass()`

RTL.h

```
#define __SVC_0   __svc_indirect(0)
extern  void rt_tsk_pass(void);
#define os_tsk_pass()  _os_tsk_pass((U32)rt_tsk_pass)
extern  void _os_tsk_pass (U32 p)   __SVC_0
```

```
LDR.W r12, [pc, #offset]
              ;Load rt_tsk_pass  in r12
SVC 0x00,
```

Generated by the compiler

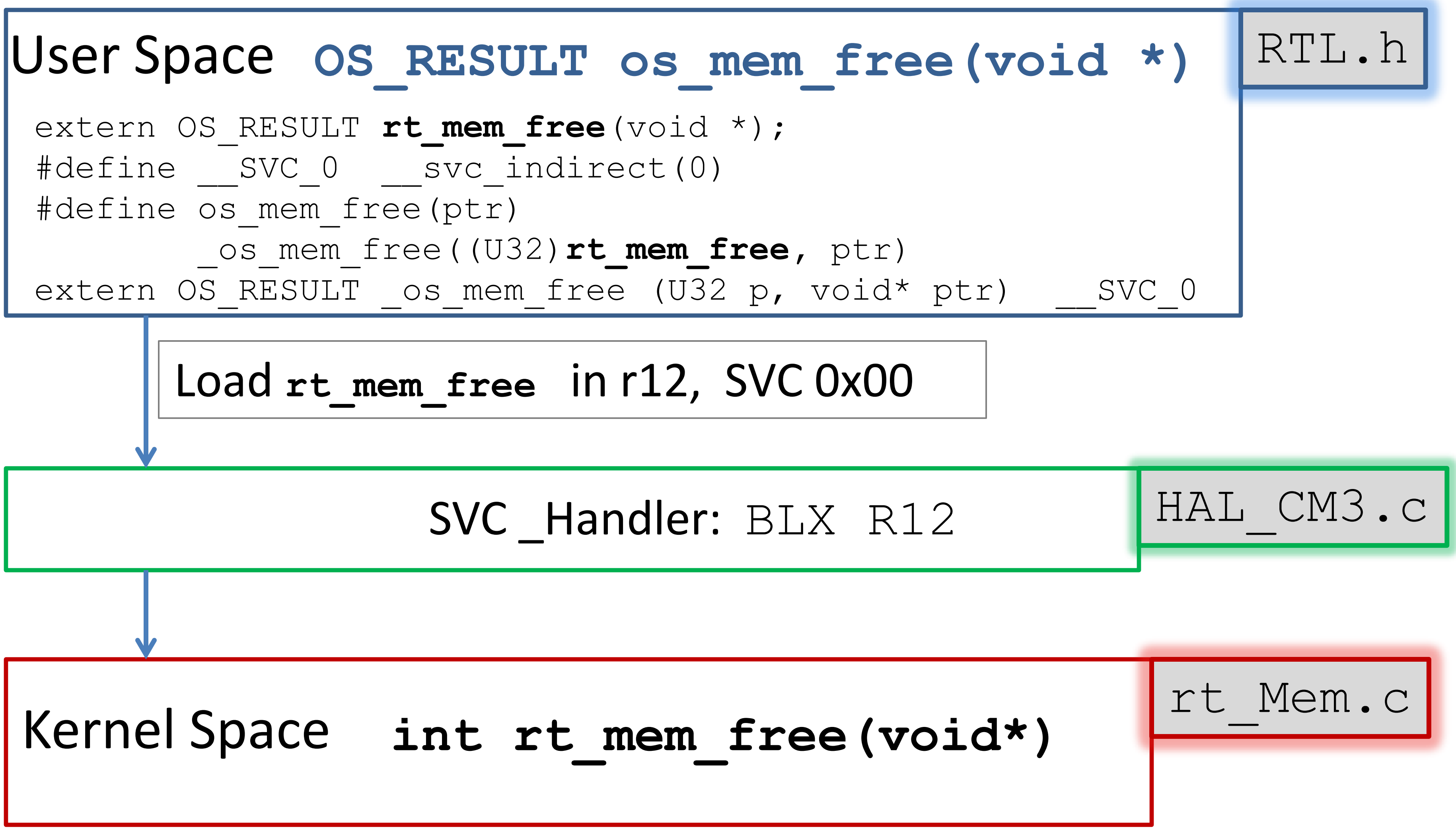
```
SVC _Handler:  BLX R12
```

HAL_CM3.c

Kernel Space `rt_tsk_pass()`

rt_Task.c

rt_Mem.c



RTL.h

User Space **`OS_RESULT os_mem_free(void *)`**

```
extern OS_RESULT rt_mem_free(void *);
#define __SVC_0  __svc_indirect(0)
#define os_mem_free(ptr)
          _os_mem_free((U32)rt_mem_free, ptr)
extern OS_RESULT _os_mem_free (U32 p, void* ptr)  __SVC_0
```

Load **`rt_mem_free`** in r12, SVC 0x00

HAL_CM3.c

SVC_Handler: `BLX R12`

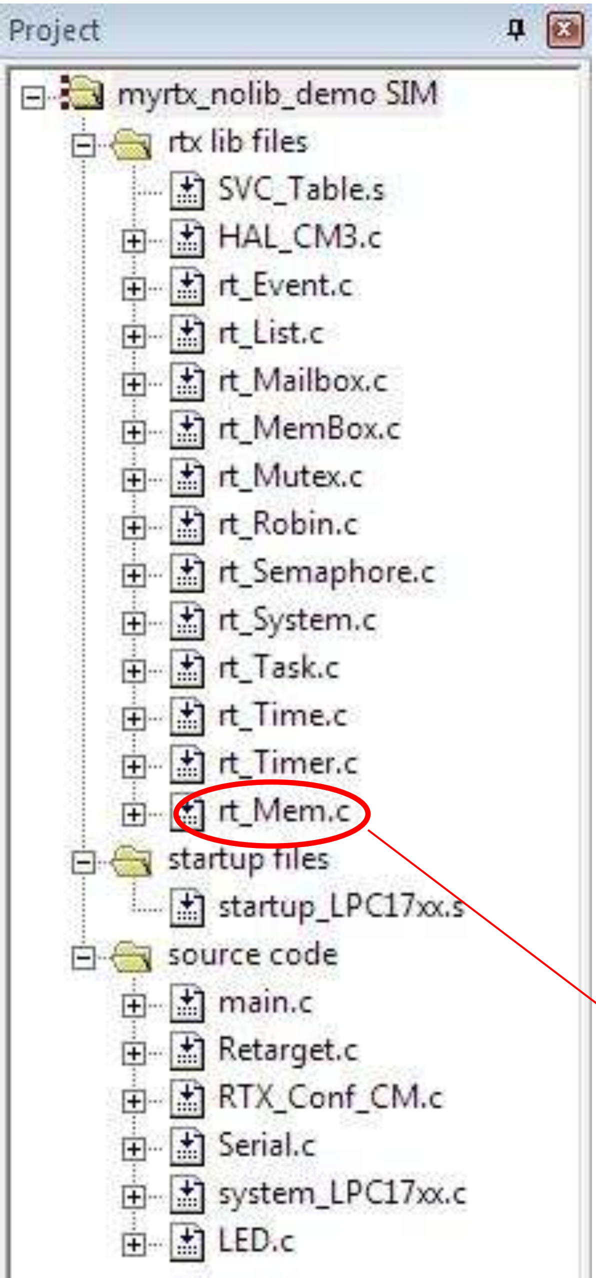
rt_Mem.c

Kernel Space **`int rt_mem_free(void*)`**

RL-RTX Kernel Files

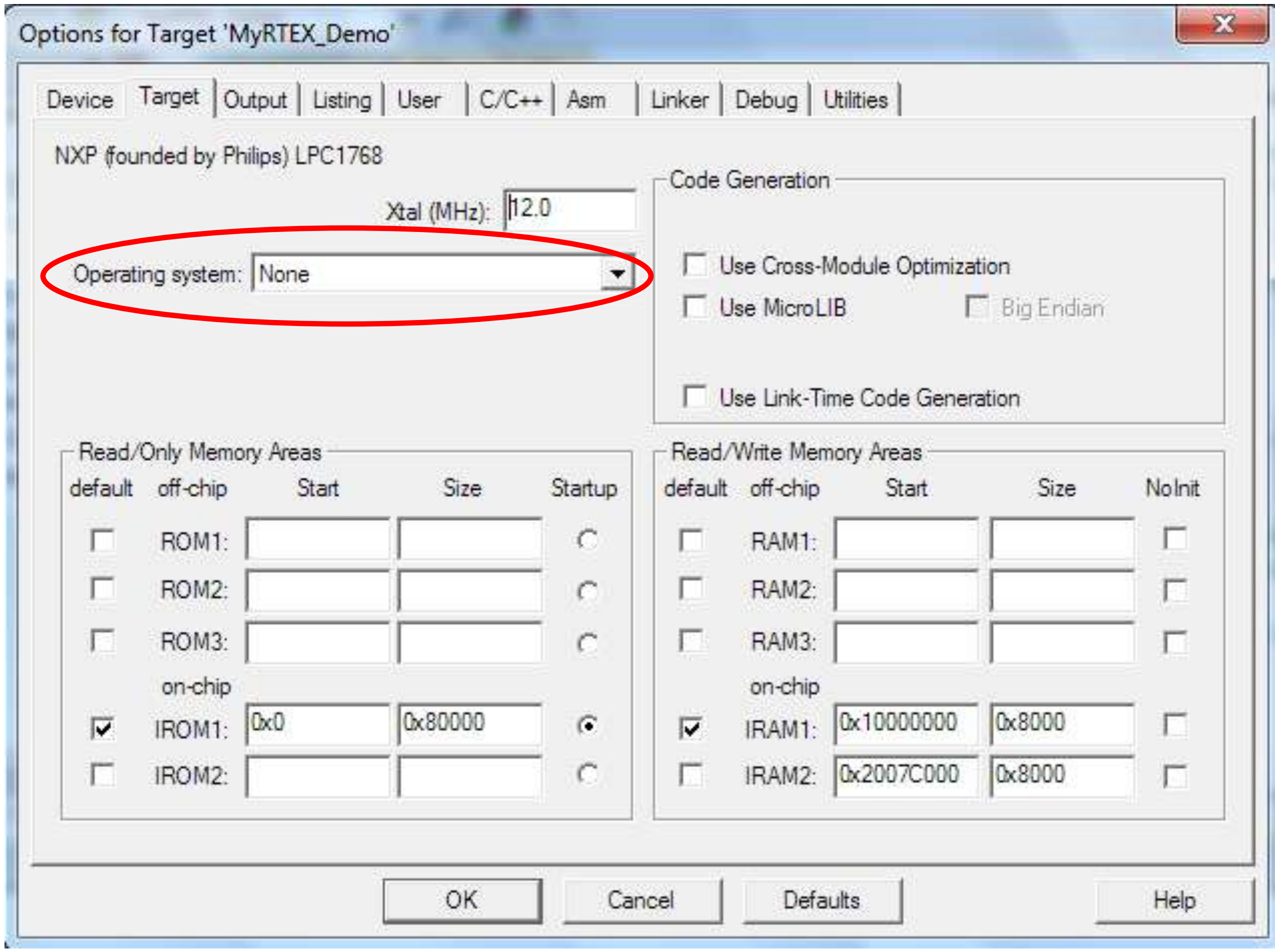
- RL-RTX Kernel Source Code
  - C:\Software\Keil\ARM\RL\RTX\SRC\CM
  - No standard C library function calls
- Add kernel files as part of your project
  - Do not add HAL_CM1.c
  - Do not add HAL_CM4.c
- Do not specify the RTX as the OS
- MicroLib is optional
- In Practice, build kernel library and link with it.

A Project with the Modified Kernel



Project

- myrtx_nolib_demo SIM
  - rtx lib files
    - SVC_Table.s
    - HAL_CM3.c
    - rt_Event.c
    - rt_List.c
    - rt_Mailbox.c
    - rt_MemBox.c
    - rt_Mutex.c
    - rt_Robin.c
    - rt_Semaphore.c
    - rt_System.c
    - rt_Task.c
    - rt_Time.c
    - rt_Timer.c
    - rt_Mem.c
  - startup files
    - startup_LPC17xx.s
  - source code
    - main.c
    - Retarget.c
    - RTX_Conf_CM.c
    - Serial.c
    - system_LPC17xx.c
    - LED.c



Options for Target 'MyRTEX_Demo'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

NXP (founded by Philips) LPC1768

Xtal (MHz): 12.0

Operating system: None

Code Generation

- [ ] Use Cross-Module Optimization
- [ ] Use MicroLIB
- [ ] Big Endian
- [ ] Use Link-Time Code Generation

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x0	0x80000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x10000000	0x8000	☐
☐	IRAM2:	0x2007C000	0x8000	☐

This is not part of the stocked RTX library.
It is a template file for lab3.

Hints

- RL-RTX Kernel
  - Kernel data structure in `rt_TypeDef.h`
  - Task management in `rt_Task.c`
    - `rt_block(U16, U8)`
    - `os_active_TCB`
    - `os_tsk`
    - `os_rdy`
    - `os_dly`
    - `os_idle_TCB` (V4.60 vs. V4.11 or older)
  - Start from `rt_MemBox.c` to see how the stocked RTX keeps track of free memory blocks.
  - Read the `rt_Mailbox.c` file `rt_mbx_send()` function to see how the return value of `rt_mbx_receive()` function is set when a task is found waiting for a message and unblocked.

References

1. Yiu, Joseph, *The Definite Guide to the ARM Cortex-M3*, 2009
2. *RealView® Compilation Tools Version 4.0 Developer Guide*
3. *ARM Software Development Toolkit Version 2.50 Reference Guide*
4. *LPC17xx User's Manual*