UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

SE350 RTX LAB 1 - 0

MCB1700 Board, Cortex-M3 Processor and System Calls

Irene Yiqing Huang
Electrical and Computer Engineering Department

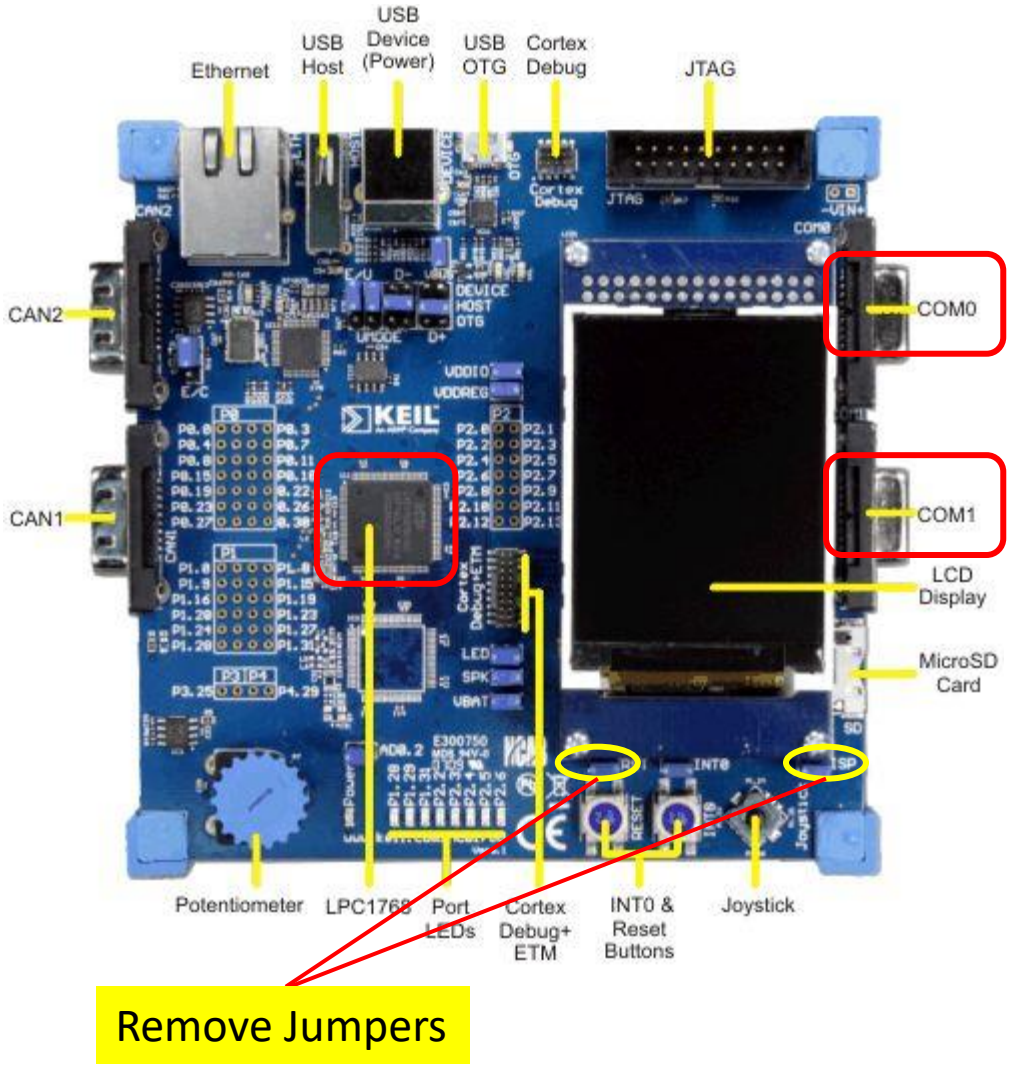
Lab Manual Reading List

Section	Topics	
10.1	MCB1700 Board Overview	Skim
10.2	Cortex-M3 Processor	Study
10.3	Memory Map	Skim
10.4	Exceptions and Interrupts	Study
9.1-9.2	The Tumb-2 ISA and AAPCS	Skim
9.3.3-9.3.4	System Exceptions and Intrinsic Functions	Skim
9.5	SVC Programming	Study

The Keil Board

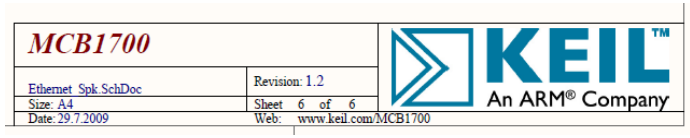
Keil MCB 1700 Board

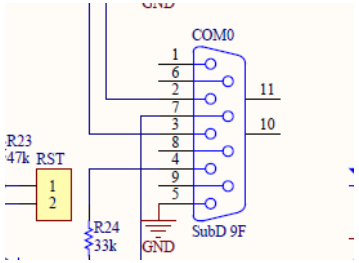
- Cortex-M3 Processor
- NXP LPC1768 MCU
- Up to 100 MHZ cpu
- One SystemTick Timer
- Four Timers
- Four UARTs
- Many other cool stuff…



Board and Chip Manuals

- MCB 1700 Schematics
- LPC17xx UM
- LPC176x Datasheet





UM10360

LPC17xx User manual

Rev. 2 — 19 August 2010 User manual

LPC1769/68/67/66/65/64/63

32-bit Arm Cortex®-M3 microcontroller; up to 512 kB flash and 64 kB SRAM with Ethernet, USB 2.0 Host/Device/OTG, CAN

Rev. 9.9 — 18 March 2020 Product data sheet

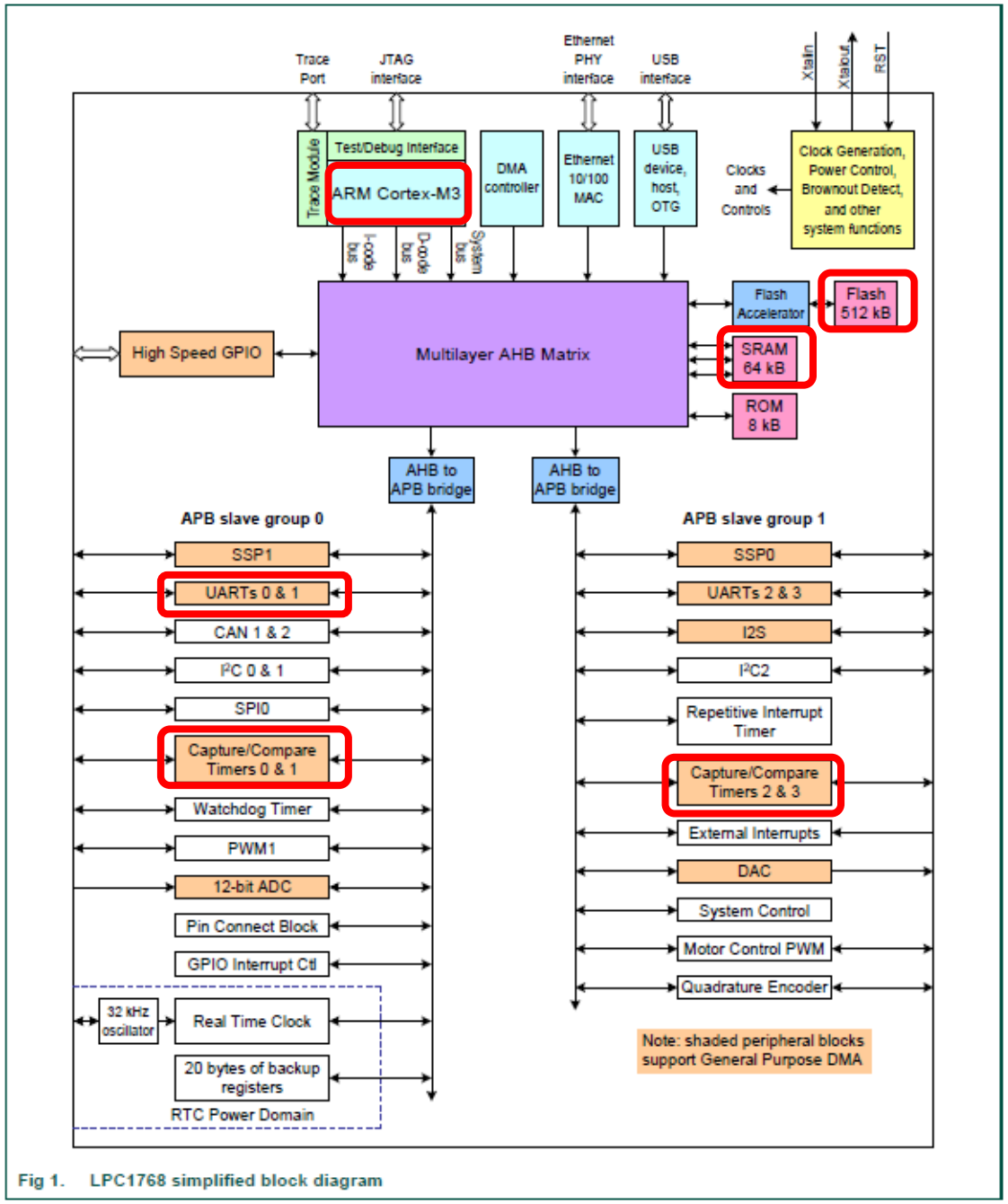


Fig 1. LPC1768 simplified block diagram

LPC17xx_UM.pdf
Page 8

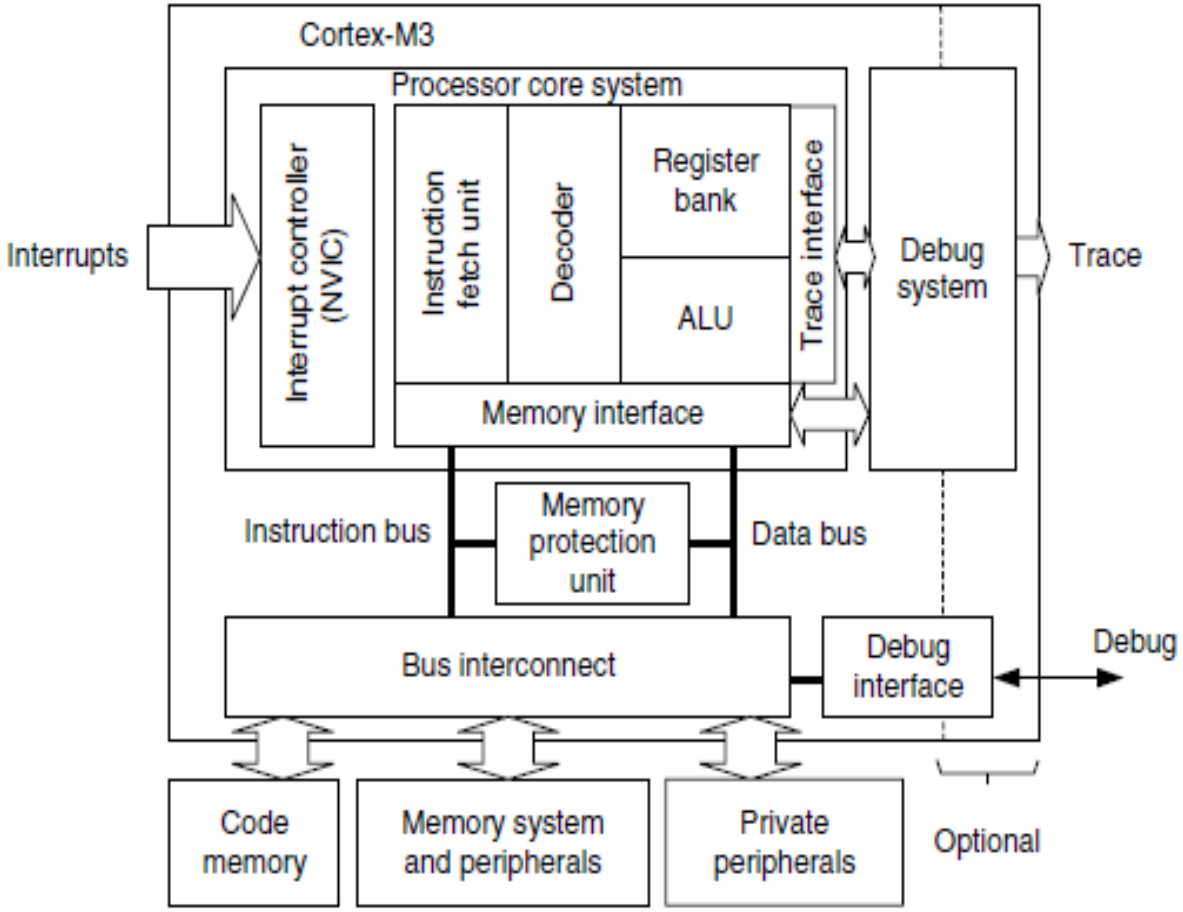
Table 3. LPC17xx memory usage and details

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
		0x0000 0000 - 0x0000 7FFF	For devices with 32 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
		0x1000 0000 - 0x1000 1FFF	For devices with 8 kB of local SRAM.
	Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each.
	AHB peripherals	0x5000 0000 - 0x501F FFFF	DMA Controller, Ethernet interface, and USB interface.
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.

LPC17xx_UM.pdf
Page 12

CORTEX-M3 Processor

Cortex-M3 Overview

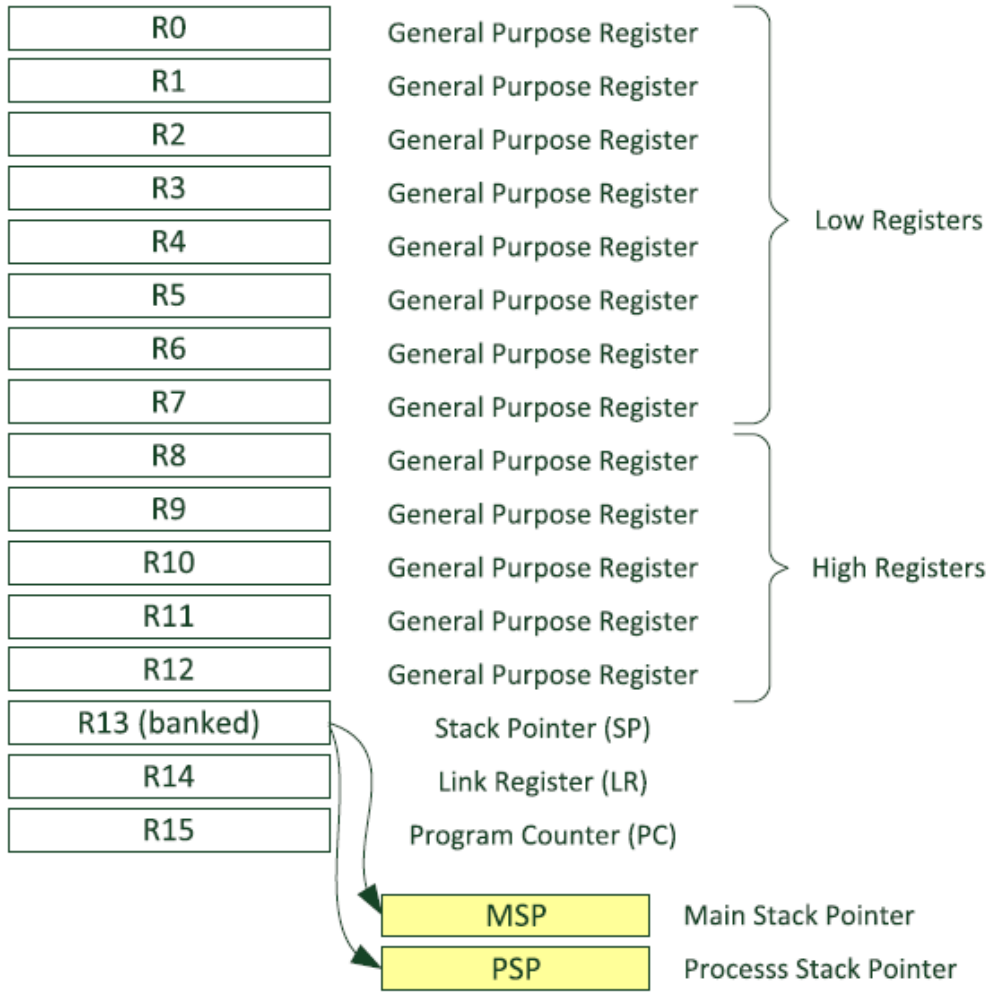


(Image Courtesy of [1])

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

Register Bank

- R0 – R7 Low registers
  - 16-bit Thumb instructions
  - 32-bit Thumb-2 instructions
- R8 – R12 High registers
  - 32-bit Thumb-2 instructions
- SP (R13) Banked Stack Pointer
  - MSP
  - PSP
- LR (R14) Link Register
- PC (R15) Program Counter



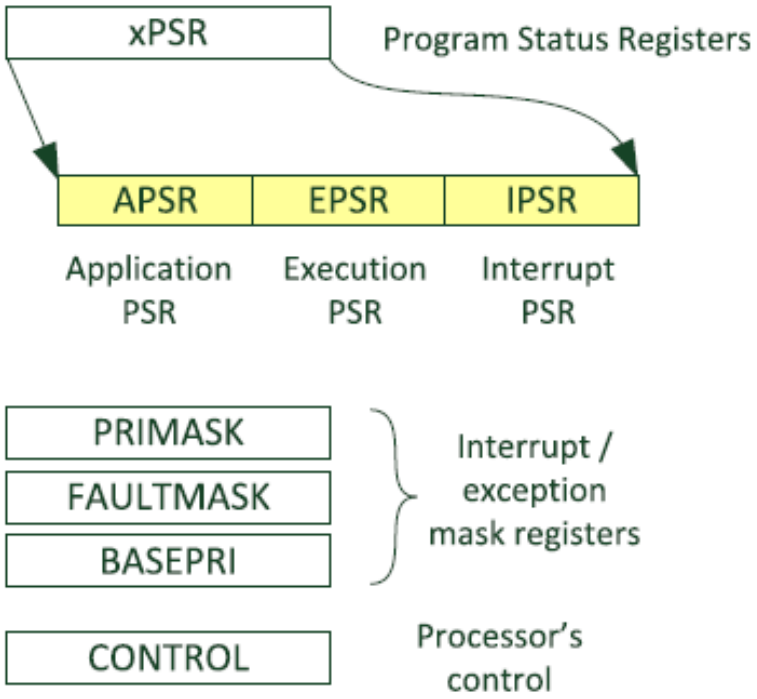
(Image Courtesy of [1])

Special Registers

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

	31:3	2	1	0
CONTROL			SPSEL	nPRIV



(Image Courtesy of [1])

The Thumb-2 ISA

- General formating of the assembler code

```
Label
        opcode operand1, operand2, …; Comments
```

Frequently Used Instructions

Mnemonic	Operands/Examples	Description
LDR	$Rt, [Rn, \#offset]$	Load Register with word
	LDR R1, [R0, #24]	Load word value from an memory address R0+24 into R1
LDM	$Rn\{!\}, reglist$	Load Multiple registers
	LDM R4, {R0 − R1}	Load word value from memory address R4 to R0, increment the address, load the value from the updated address to R1.
STR	$Rt, [Rn, \#offset]$	Store Register word
	STR R3, [R2, R6]	Store word in R3 to memory address R2+R6
	STR R1, [SP, #20]	Store word in R1 to memory address SP+20
MRS	$Rd, spec\_reg$	Move from special register to general register
	MRS R0, MSP	Read MSP into R0
	MRS R0, PSP	Read PSP into R0
MSR	$spec\_reg, Rm$	Move from general register to special register
	MSR MSP, R0	Write R0 to MSP
	MSR PSP, R0	Write R0 to PSP
PUSH	$reglist$	Push registers onto stack
	PUSH {R4 − R11, LR}	push in order of decreasing the register numbers
POP	$reglist$	Pop registers from stack
	POP {R4 − R11, PC}	pop in order of increasing the register numbers
BL	$label$	Branch with Link
	BL funC	Branch to address labeled by funC, return address stored in LR
BLX	Rm	Branch indirect with link
	BLX R12	Branch with link and exchange (Call) to an address stored in R12
BX	Rm	Branch indirect
	BX LR	Branch to address in LR, normally for function call return

Table 9.1: Assembler instruction examples

CMSIS Structure

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers
  - NVIC, MPU, System Control Block, SysTick registers
  - core_cm*[ch] files
- Standardized system exception names. For example:

```
void SVC_Handler();
void UART0_IRQHandler();
```

- Standardized method of header file organization
- Common method for system initialization

```
SystemInit()
```

- Standardized intrinsic functions. For example:

```
void __disable_irq(void);
void enable_irq(void);
```

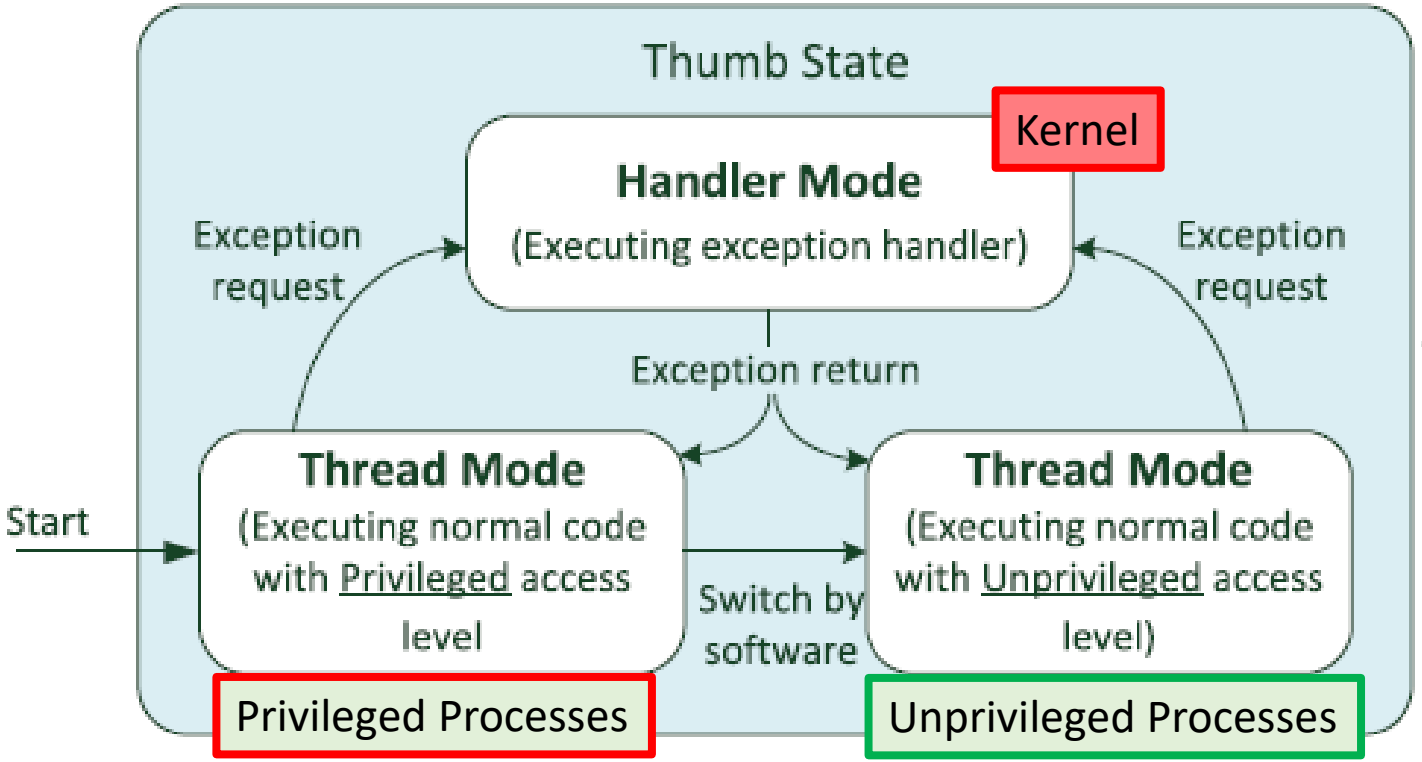
- Standardized way for embedded software to determine system clock frequency
  - `SystemFrequency` variable is defined in device driver code
- Vendor peripherals with standardized C structure

CMSIS Functions

Registers	Access	CMSIS Function	Instruction
PRIMASK FAULTMASK	Write	`void __enable_irq(void)`	`CPSIE I`
	Write	`void __disable_irq(void)`	`CPSID I`
CONTROL	Read	`uint32_t __get_CONTROL(void)`	`MRS R0, CONTROL`
	Write	`void __set_CONTROL(uint32_t value)`	`MSR CONTROL, R0`
MSP	Read	`uint32_t __get_MSP(void)`	`MRS R0, MSP`
	Write	`void __set_MSP(uint32_t value)`	`MSR MSP, R0`
PSP	Read	`uint32_t __get_PSP(void)`	`MRS R0, PSP`
	Write	`void __set_PSP(uint32_t value)`	`MSR PSP, R0`

Operation Modes

- Two modes
  - Thread mode
  - Handler mode
- Two access levels
  - Unprivileged/User level
  - Privileged level



(Image Courtesy of [1])

Operation Modes and Stacks

Processor Mode	Stack used	CONTROL		Privilege level	Space
		Bit[1]	Bit[0]		
Handler	MSP	0 (RO)	-	Privileged	Kernel
Thread	MSP	0	0	Privileged	Kernel-
	PSP	1	0	Privileged	Kernel-
	PSP	1	1	Unprivileged	User
	MSP	0	1	Unprivileged	User

```
/* Assembly Instructions */
MSR CONTROL, R0 ; write to CONTROL
MRS R0, CONTROL ; read from CONTROL

/* CMSIS functions */
__set_CONTROL(uint32_t)  /* write to CONTROL */
__get_CONTROL()          /* read from CONTROL */
```

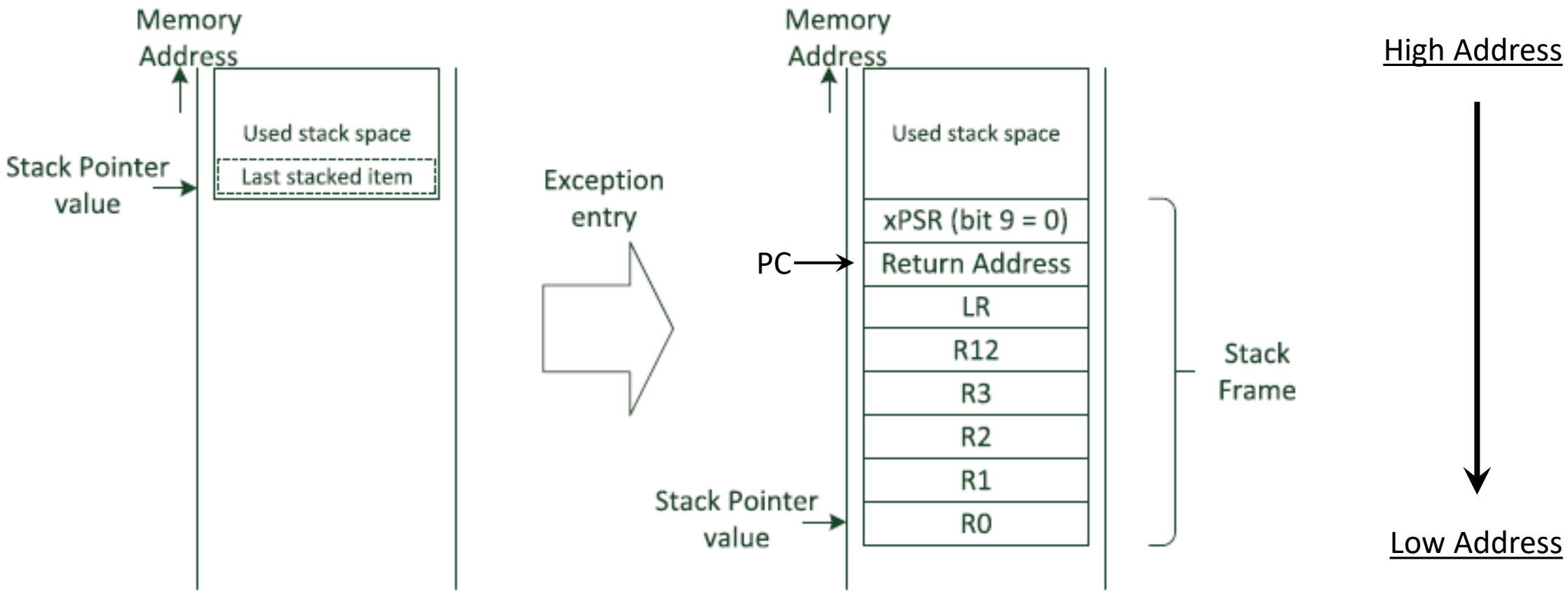
Exceptions

- Nested Vectored Interrupt Controller (NVIC)
  - System Exceptions
    - Exception Numbers 1 -15
    - SVC call exception number is 11 (LAB1)
  - External Interrupts
    - Exception Numbers 16-50
    - Timer0-3 IRQ numbers are 17-20 (LAB 2)
    - UART0-3 IRQ numbers are 21-24 (LAB 3)
- Vector table is at 0x0 after reset.
- 32 programmable priorities
- Each vector table entry contains the exception handler's address (i.e. entry point)

Exception Entry

- When
  - The processor is in thread mode
  - The new exception is of higher priority than the exception being handled
- Stacking the exception stack frame
  - PSR, PC, LR, R12, R0-R3
  - SP is updated
  - SP switches to MSP if it was pointing to PSP
- Fetch Exception Handler Entry Address from Vector Table
- Writes EXC_RETURN value to LR
  - Which stack pointer corresponds to the exception stack frame
  - What operation mode the processor was in before exception entry

Exception Stack Frame



When double-word stack alignment adjustment is not required

(Image Courtesy of [1])

Exception Return

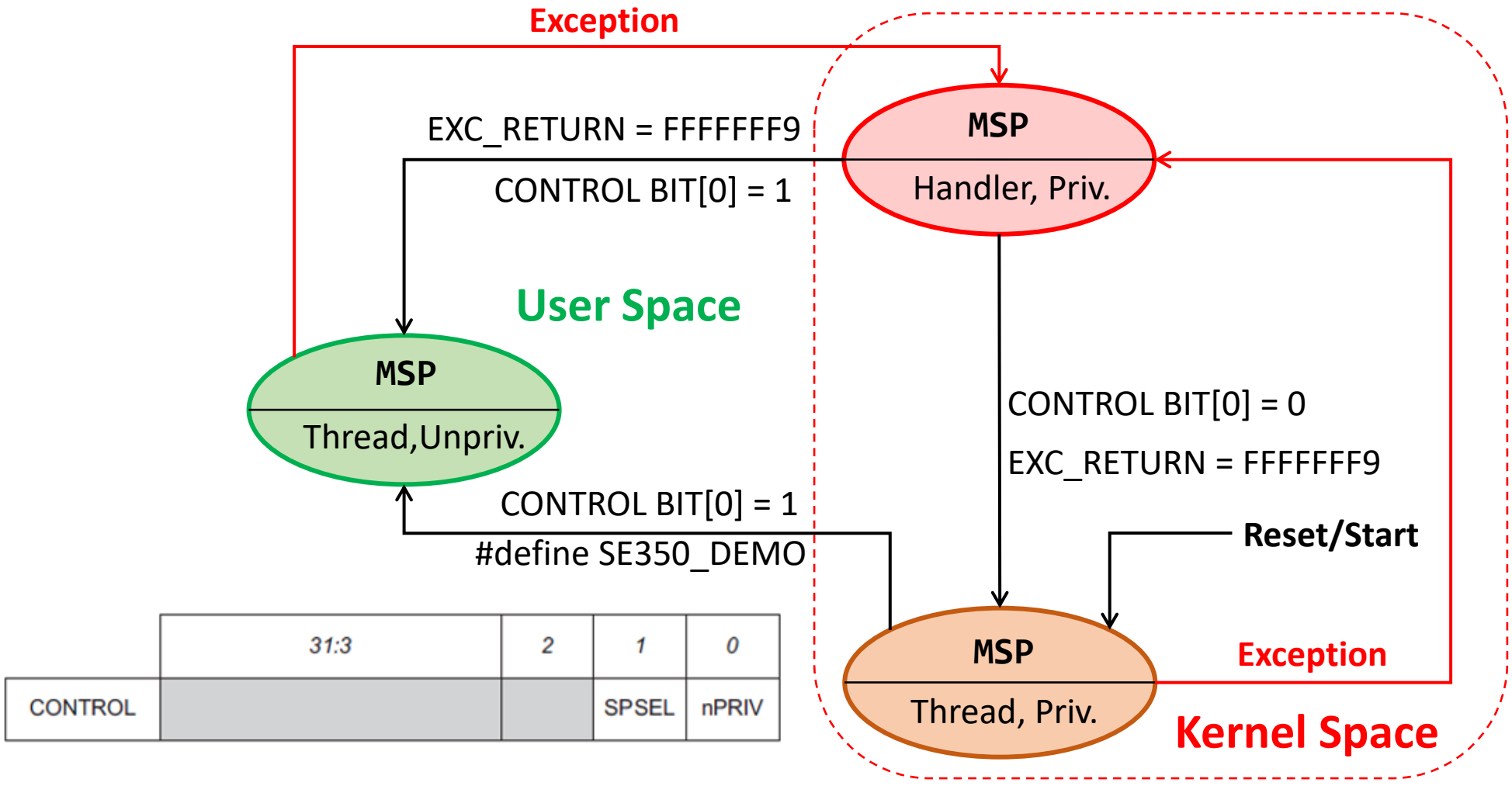
- The EXC_RETURN values

Value	Description		
	Return Mode	Exception return gets state from	SP after return
0xFFFFFFF1	Handler	MSP	MSP
0xFFFFFFF9	Thread	MSP	MSP
0xFFFFFFFD	Thread	PSP	PSP

we use this in lab

- Unstacking exception stack frame
- NVIC register update
  - Clear active bit of the exception
  - Pending bit sets if the external interrupt is still asserted.

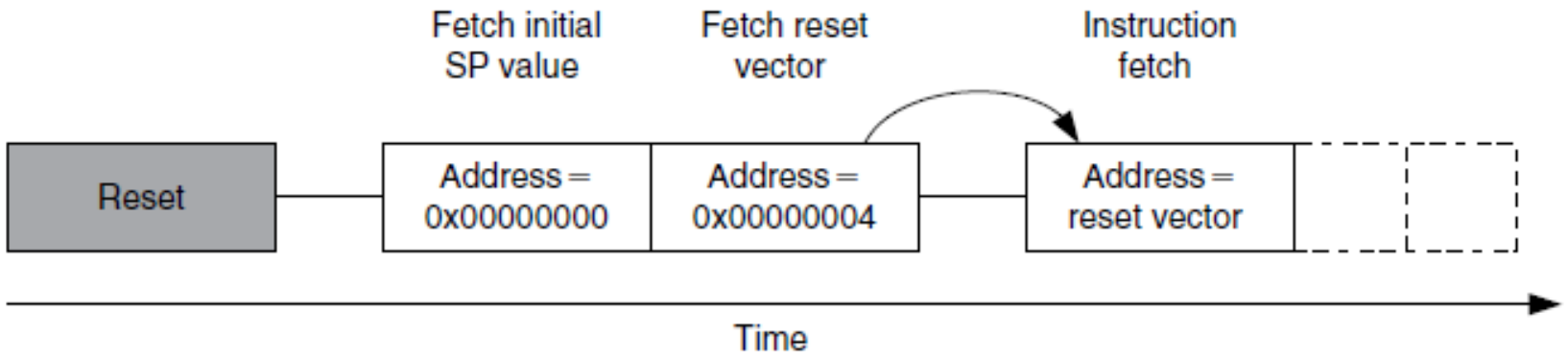
Starter Code Modes and Stacks Diagram



	31:3	2	1	0
CONTROL			SPSEL	nPRIV

Reset Sequence

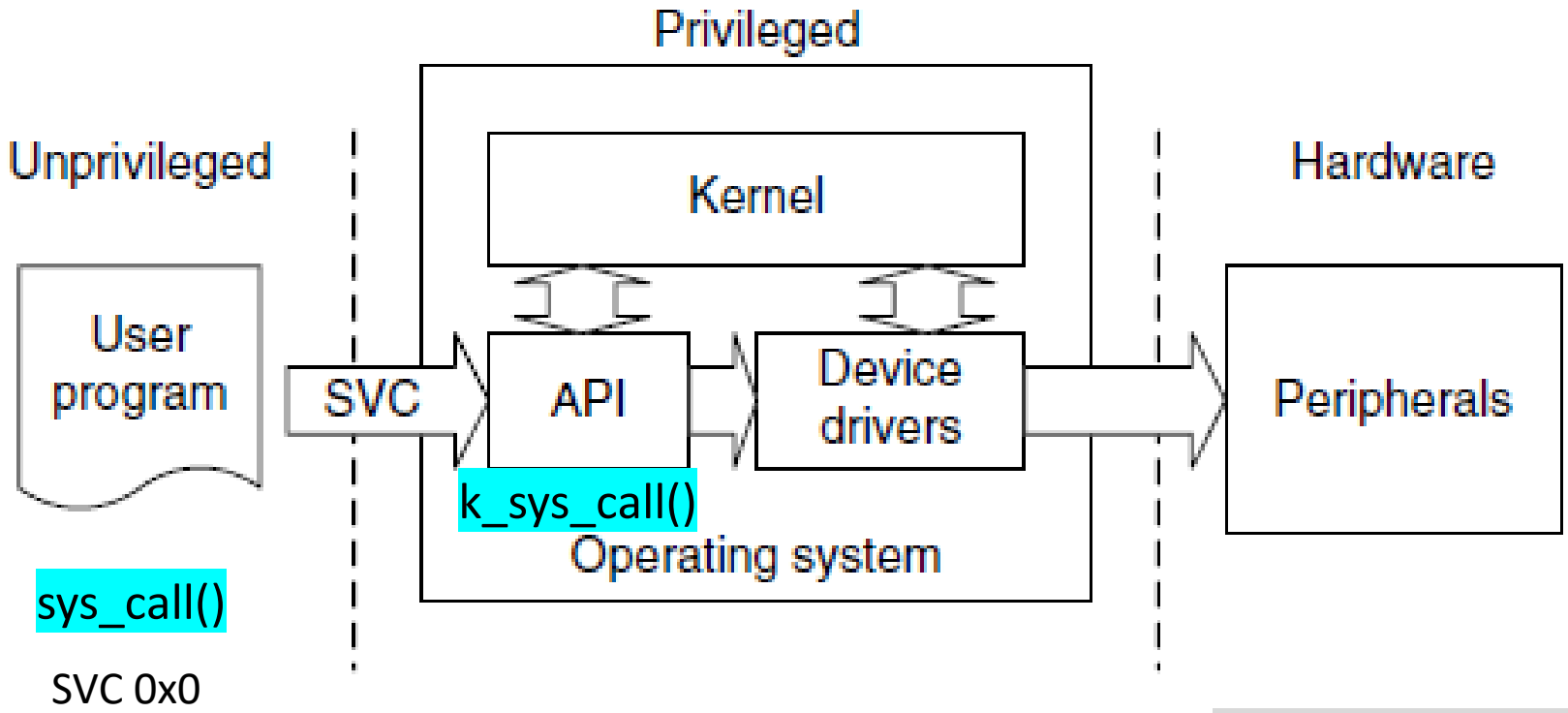
Address	Exception Number	Value (Word Size)
0x0000 0000	-	MSP initial value
0x0000 0004	1	Reset vector (program counter initial value)
0x0000 0008	2	NMI handler starting address
0x0000 000C	3	Hard fault handler starting address
…	…	Other handler starting address



(Image Courtesy of [1])

SVC and System Calls

SVC as a Gateway for OS Functions

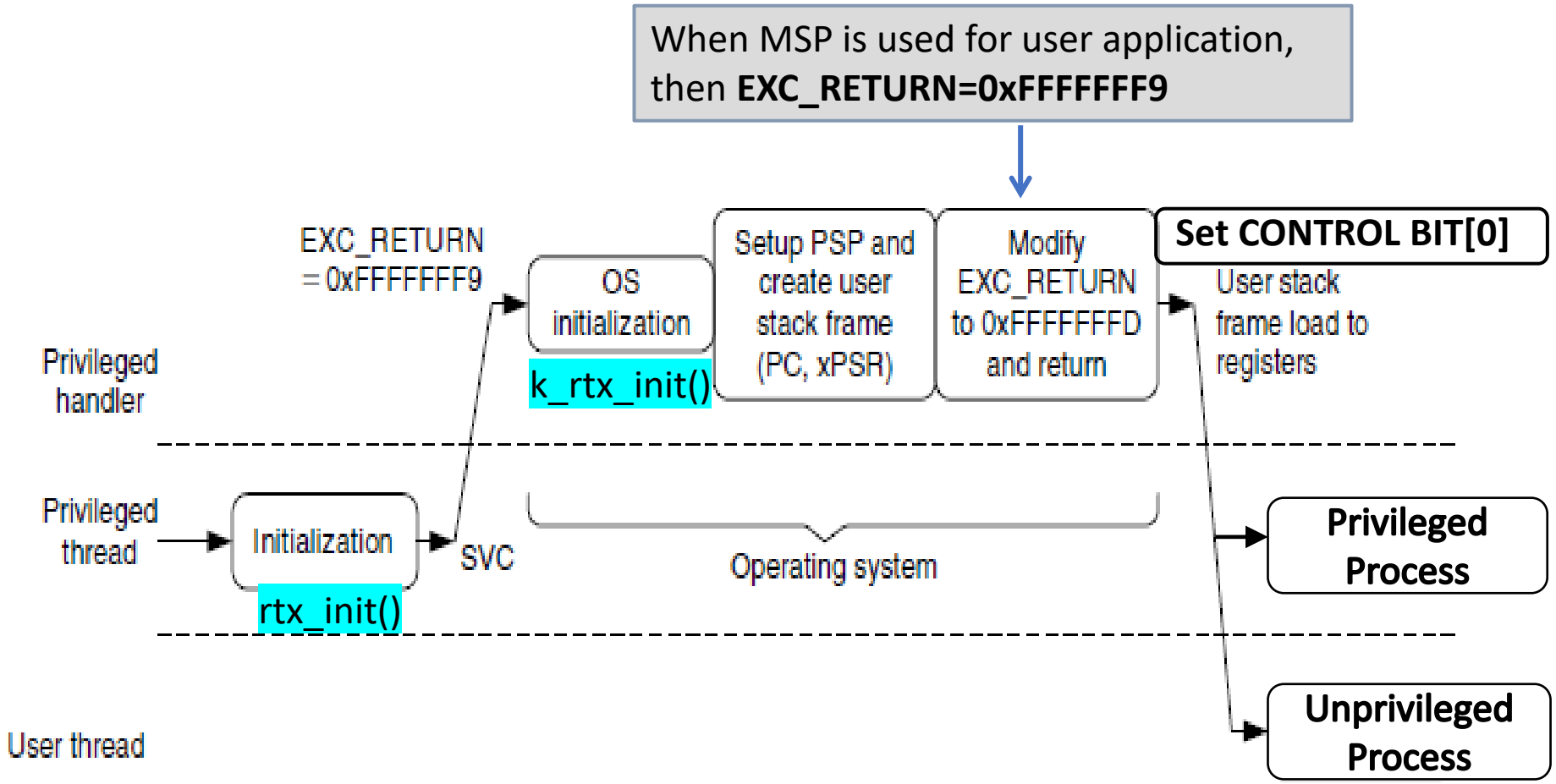


Provides the kernel with service arguments

Example:
rtx_init -> SVC 0x0 -> k_rtx_init

(Image Courtesy of [1])

OS Initialization Mode Switch



(Modified Image Courtesy of [1])

CMSIS Structure

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers
  - NVIC, MPU, System Control Block, SysTick registers
  - core_cm*[ch] files
- Standardized system exception names. For example:

```
void SVC_Handler();
void UART0_IRQHandler();
```

- Standardized method of header file organization
- Common method for system initialization

```
SystemInit()
```

- Standardized intrinsic functions. For example:

```
void __disable_irq(void);
void enable_irq(void);
```

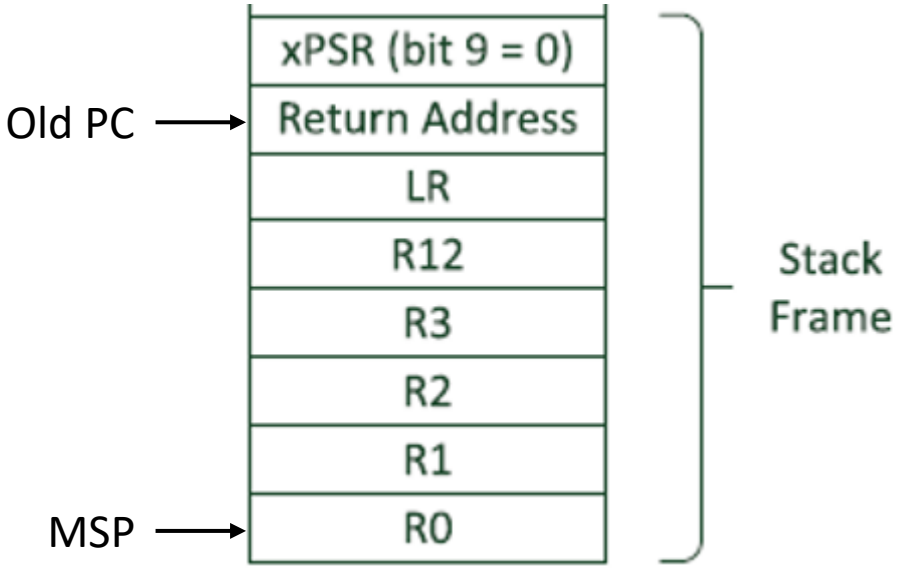
- Standardized way for embedded software to determine system clock frequency
  - `SystemFrequency` variable is defined in device driver code
- Vendor peripherals with standardized C structure

AAPCS Base Procedure Call Standard (ARM Architecture Procedure Call Standard)

- Assumptions
  - Function call input parameters and return value data types are not floating point or a data type that is bigger than 32-bit
- R0-R3, caller saved registers
  - Input parameters Px of a function. R0=P1, R1=P2, R2=P3 and R3=P4
  - **R0** is used for **return value** of a function
- R4-R11, callee saved registers
  - Must be preserved by the called function. C compiler generates push and pop assembly instructions to save and restore them automatically.
- R12, SP, LR and PC (R12-R15)
  - R12 is the Intra-Procedure-Call scratch register.
  - We do not need to save it, but we may save it for stack alignment purpose
  - LR needs to be saved by callee for return

SVC Handler in HAL.c

```
__asm void SVC_Handler(void)
{
    MRS R0, MSP

    ; extract SVC number from stacked PC

    LDM  R0, {R0-R3, R12}

    PUSH {R4-R11, LR}

    BLX  R12

    POP   {R4-R11, LR}
    ; get C function return value
    BX LR
}
```



System Calls in ASM

rtx.h

```
int release_memory_block(void *);
```

User Space

HAL.c

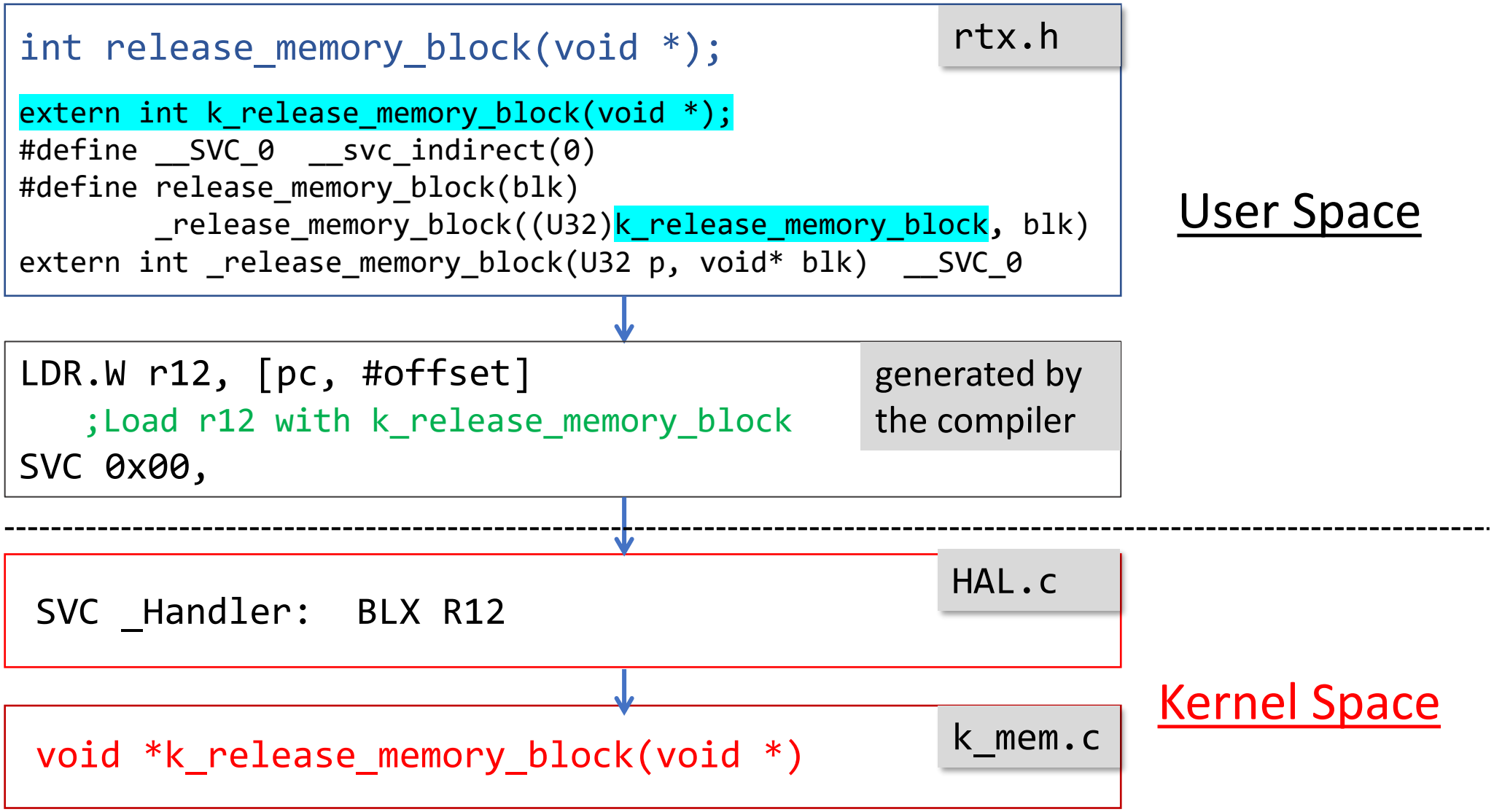
```
__asm int release_memory_block(void *) {
  ;load k_release_memory_block  in r12
  LDR.W r12,=__cpp(k_release_memory_block)
  ;code fragment omitted
  SVC 0x00
  BX LR
  ALIGN
}
SVC_Handler:  BLX R12
```

Kernel Space

k_mem.c

```
int k_release_memory_block(void *)
```

System Calls in C



UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Thank you!

Electrical and Computer Engineering Department