Interrupts (1)

Lecture 9

Yeongpil Cho

Hanynag University

Topics

- Interrupt Handling Basics
- Priority Management

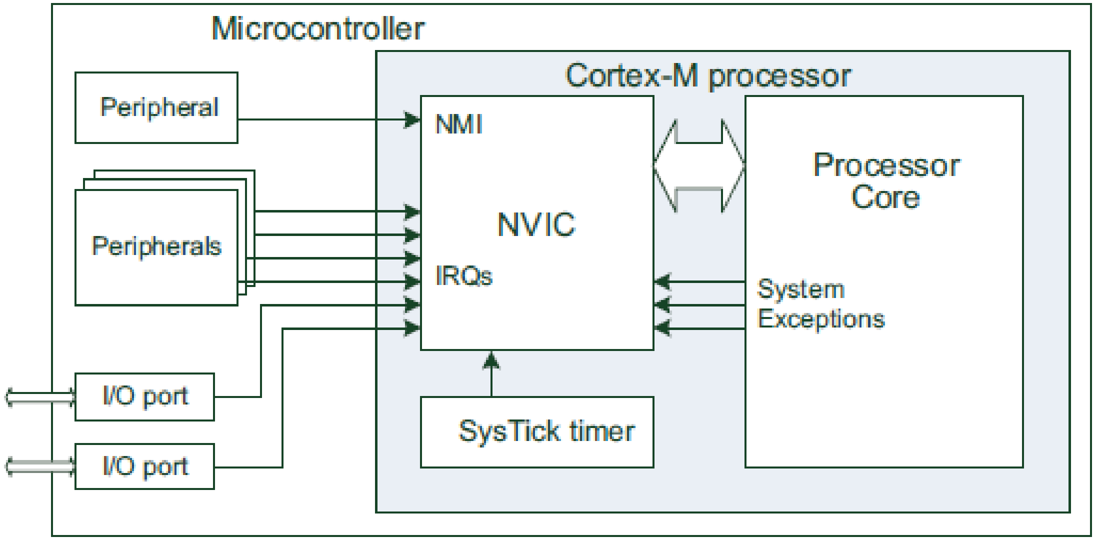
Interrupt Handling Basics

Interrupts

- Motivation
  - Inform processors of some external events timely
- Polling:
  - You pick up the phone every three seconds to check whether you are getting a call.
- Interrupt:
  - Do whatever you should do and pick up the phone when it rings.

ARMv7-M Interrupt Handling

- One Non-Maskable Interrupt (NMI) supported
- Up to 511 (496 external and 15 internal ones) prioritizable interrupts/exceptions supported
    - Interrupts can be masked
    - Implementation option selects number of interrupts supported
- Nested Vectored Interrupt Controller (NVIC) is tightly co upled with processor core

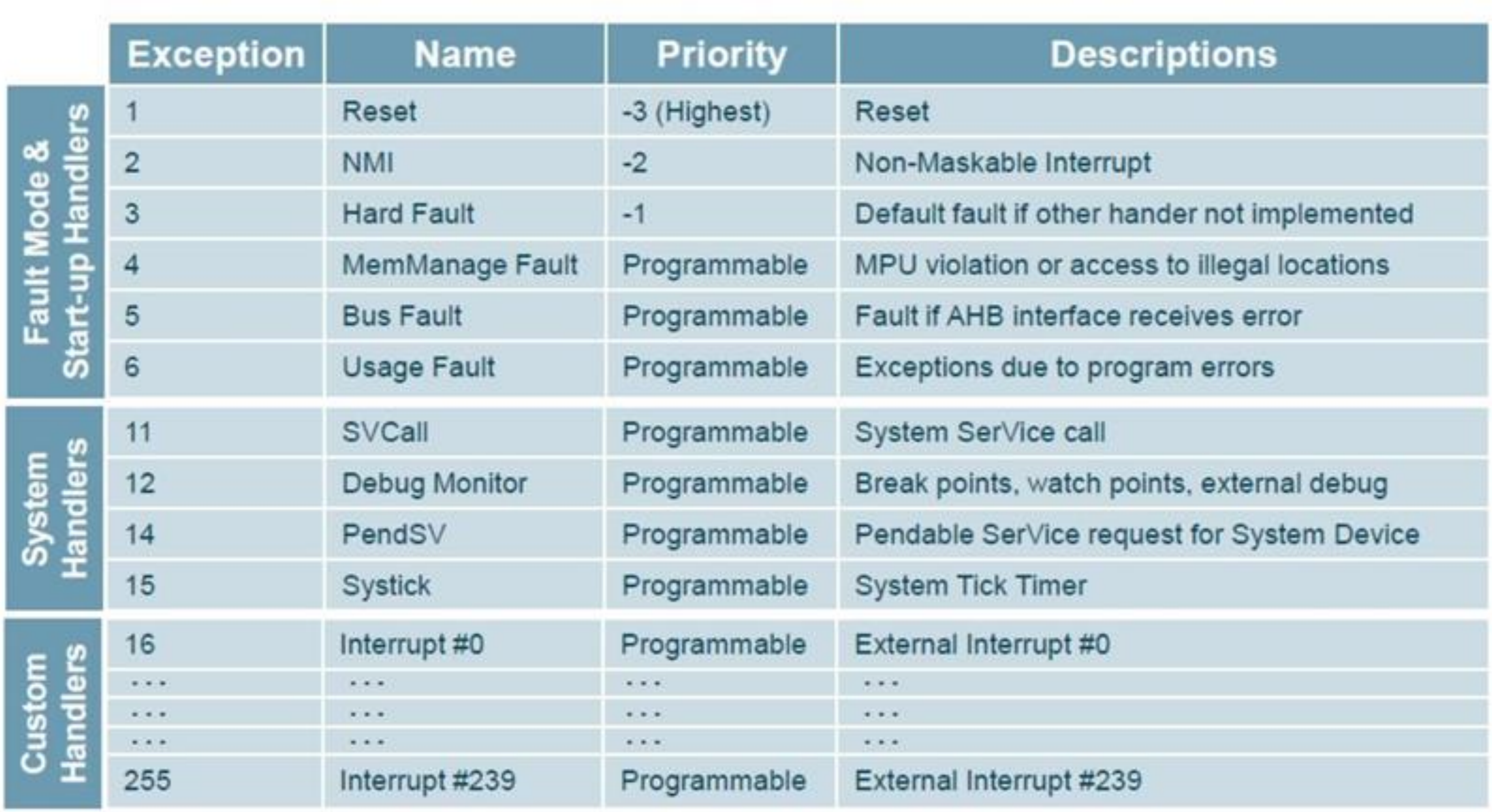


Interrupt Service Routine Vector Table

- First entry contains initial Main SP
- All other entries are addresses for
    - exception/interrupt handlers
    - Must always have LSBit = 1 (for Thumb)
- Table can be relocated
    - Use Vector Table Offset Register
    - Still require minimal table entries at 0x0 for booting the core
- Table can be generated using C code

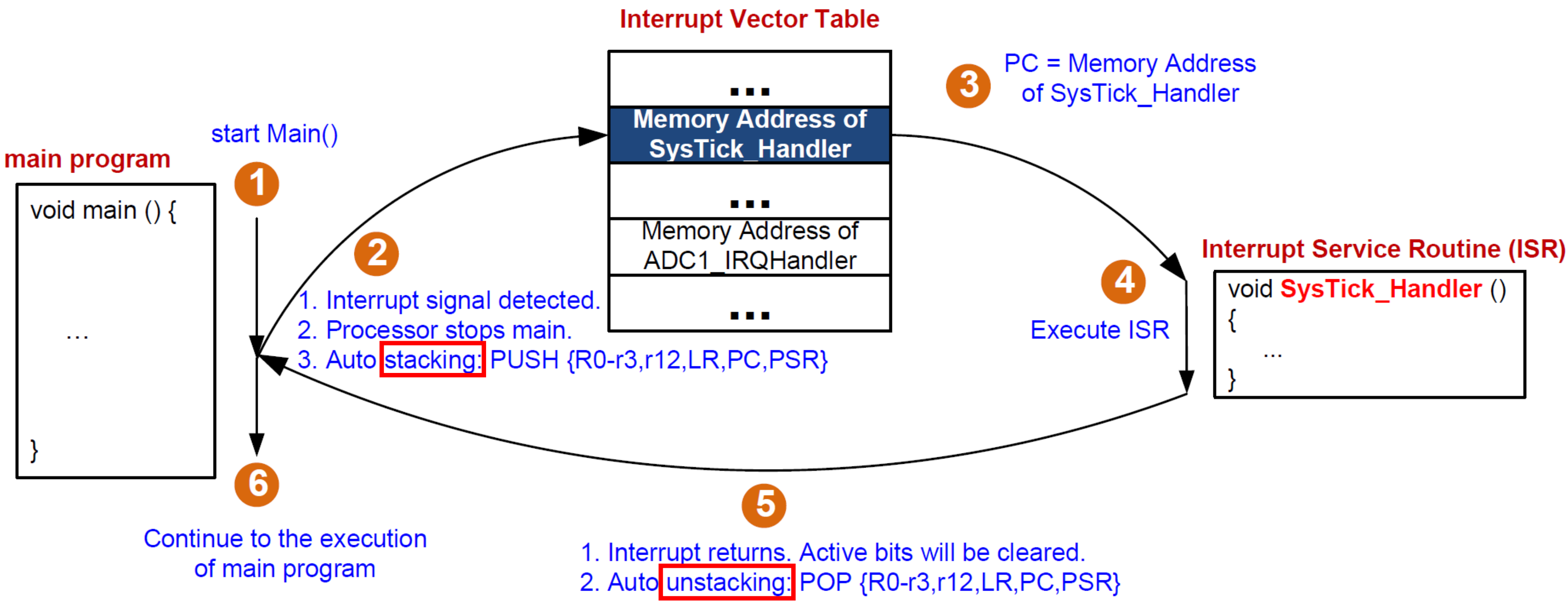
Address	
0x40 + 4*N	**External N**
…	**…**
0x40	**External 0**
0x3C	**SysTick**
0x38	**PendSV**
0x34	**Reserved**
0x30	**Debug Monitor**
0x2C	**SVC**
0x1C to 0x28	**Reserved (x4)**
0x18	**Usage Fault**
0x14	**Bus Fault**
0x10	**Mem Manage Fault**
0x0C	**Hard Fault**
0x08	**NMI**
0x04	**Reset**
0x00	**Initial Main SP**

Interrupt Service Routine Vector Table of Cortex-M processors



	Exception	Name	Priority	Descriptions
Fault Mode & Start-up Handlers	1	Reset	-3 (Highest)	Reset
	2	NMI	-2	Non-Maskable Interrupt
	3	Hard Fault	-1	Default fault if other hander not implemented
	4	MemManage Fault	Programmable	MPU violation or access to illegal locations
	5	Bus Fault	Programmable	Fault if AHB interface receives error
	6	Usage Fault	Programmable	Exceptions due to program errors
System Handlers	11	SVCall	Programmable	System SerVice call
	12	Debug Monitor	Programmable	Break points, watch points, external debug
	14	PendSV	Programmable	Pendable SerVice request for System Device
	15	Systick	Programmable	System Tick Timer
Custom Handlers	16	Interrupt #0	Programmable	External Interrupt #0
	...	...	...	...
	...	...	...	...
	...	...	...	...
	255	Interrupt #239	Programmable	External Interrupt #239

Interrupt Handling Process



Interrupt Vector Table

...

Memory Address of SysTick_Handler

...

Memory Address of ADC1_IRQHandler

...

main program

```
void main () {

    ...

}
```

1 start Main()

2
1. Interrupt signal detected.
2. Processor stops main.
3. Auto stacking: PUSH {R0-r3,r12,LR,PC,PSR}

3 PC = Memory Address of SysTick_Handler

Interrupt Service Routine (ISR)

```
void SysTick_Handler ()
{
    ...
}
```

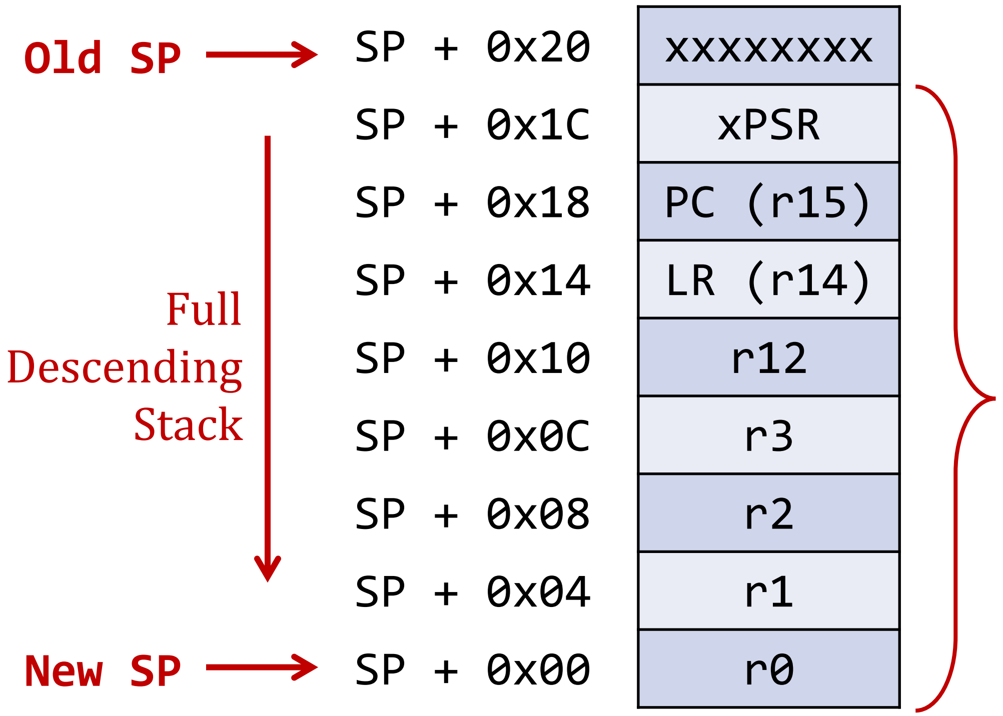
4 Execute ISR

5
1. Interrupt returns. Active bits will be cleared.
2. Auto unstacking: POP {R0-r3,r12,LR,PC,PSR}

6 Continue to the execution of main program

Stacking & Unstacking

- Current mode (either Thread mode or Handler mode)'s stack is used for stacking/unstacking.



Old SP →

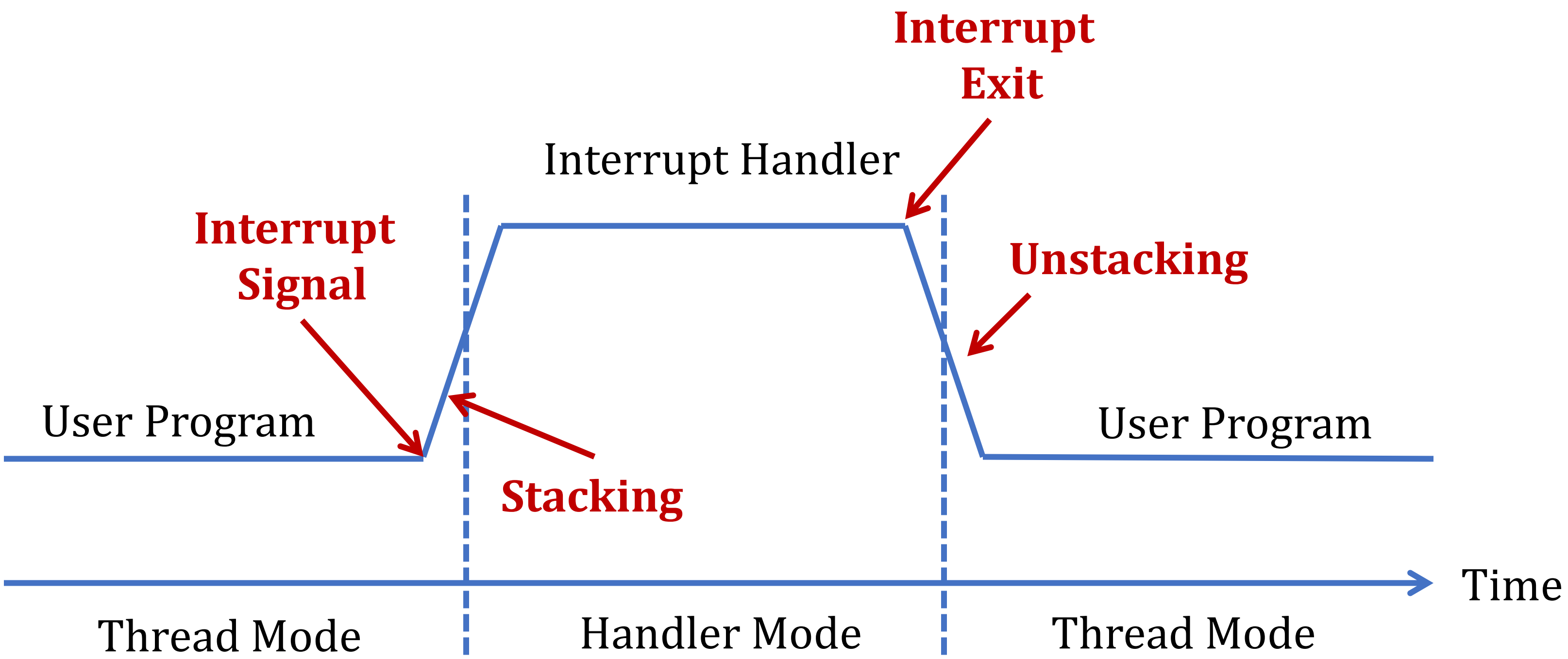
Full Descending Stack

New SP →

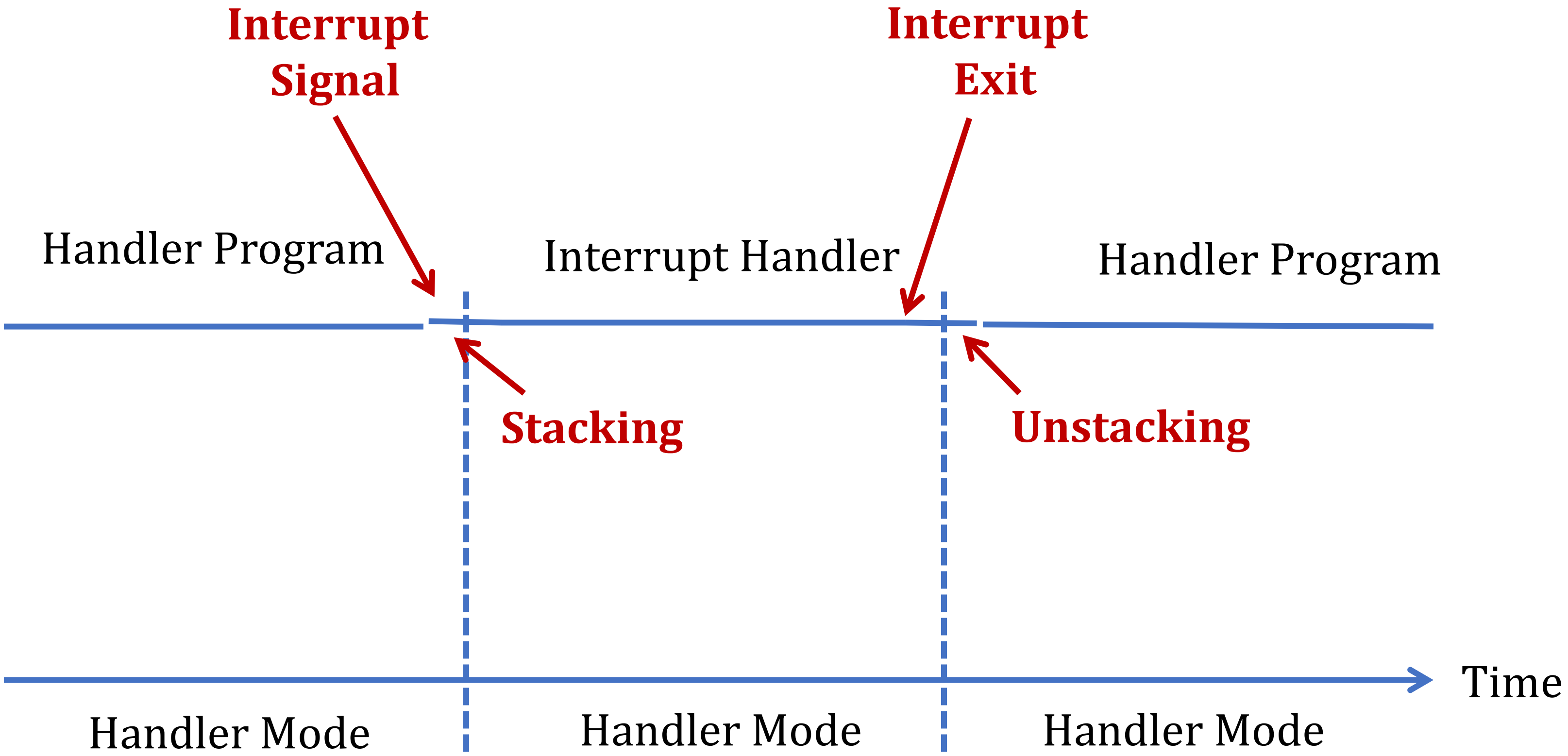
SP + 0x20	xxxxxxxx
SP + 0x1C	xPSR
SP + 0x18	PC (r15)
SP + 0x14	LR (r14)
SP + 0x10	r12
SP + 0x0C	r3
SP + 0x08	r2
SP + 0x04	r1
SP + 0x00	r0

- **Stacking**: The processor automatically pushes these eight registers into the stack before an interrupt handler starts
- **Unstacking:** The processor automatically pops these eight register out of the stack when an interrupt hander exits.

Stacking & Unstacking

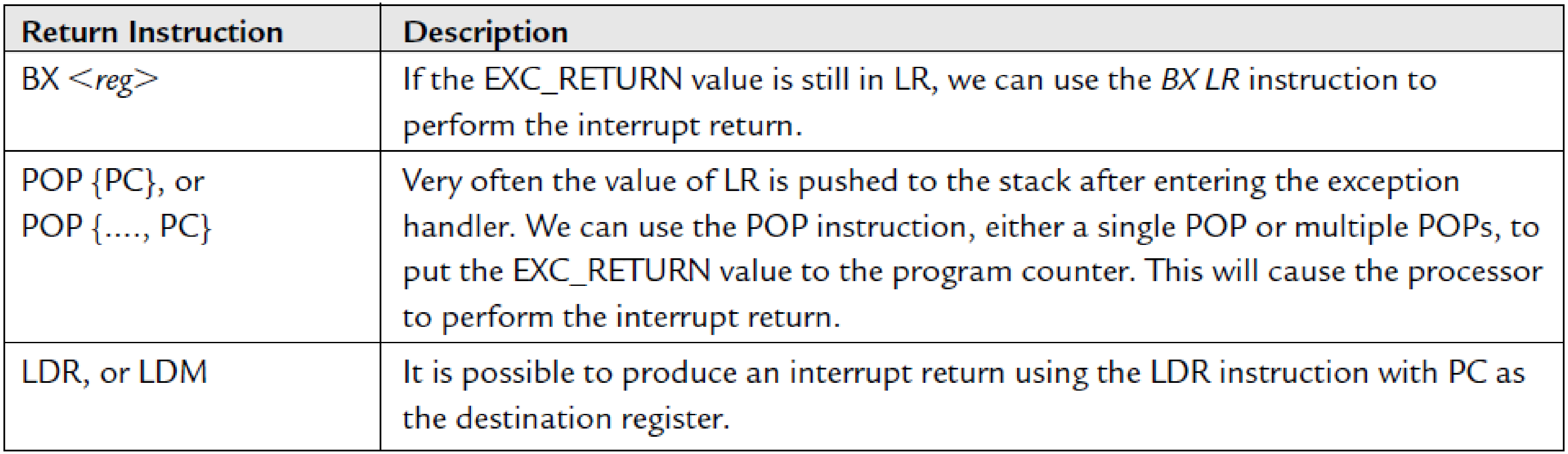


Stacking & Unstacking



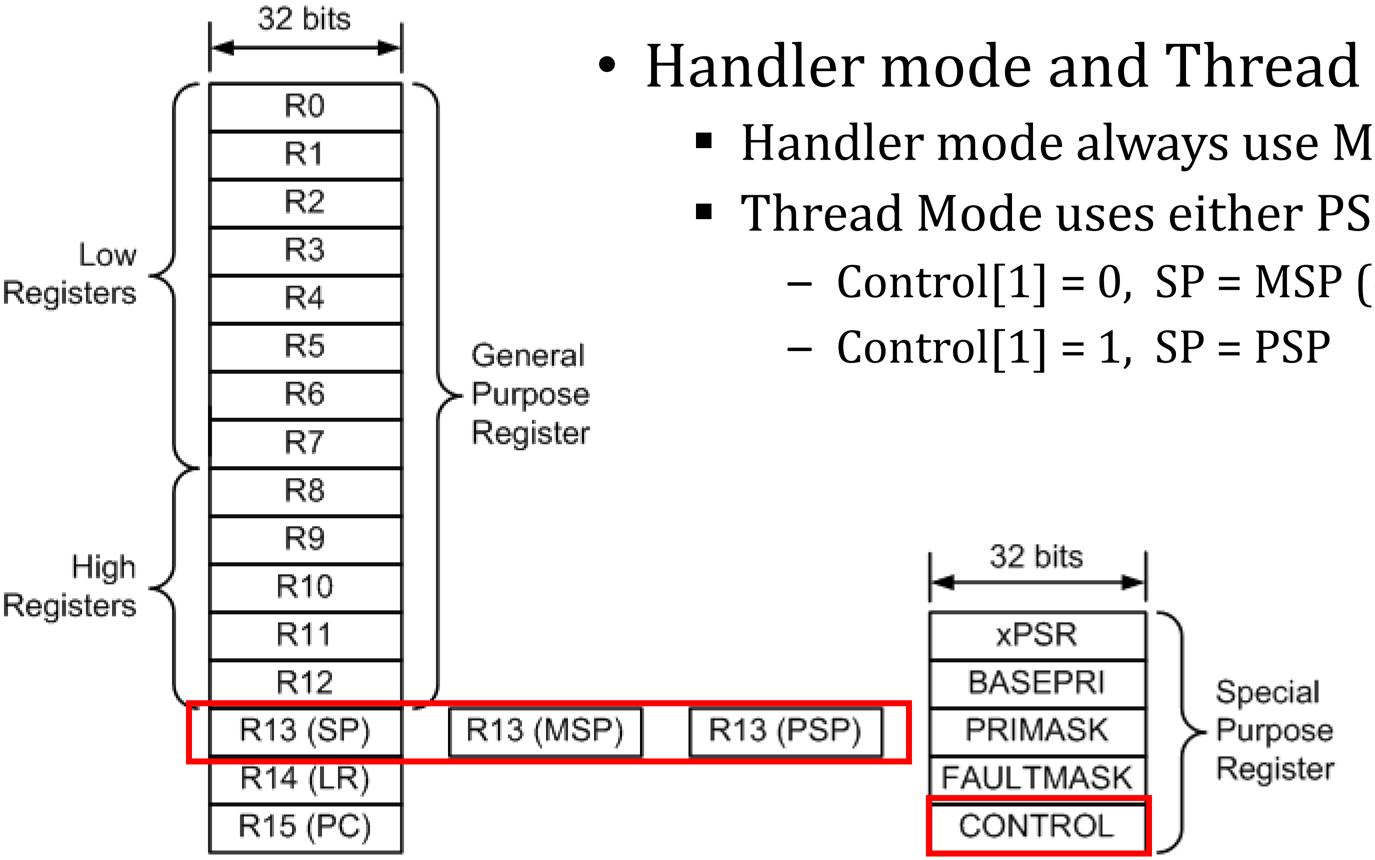
Exception Exits

- When the **EXC_RETURN** is loaded into the PC at the end of the exception handler execution, the processor performs an exception return sequence
- EXC_RETURN is generated and set to **LR** by processors when an exception arises.
- There are three ways to trigger the interrupt return sequence:



Return Instruction	Description
BX <*reg*>	If the EXC_RETURN value is still in LR, we can use the *BX LR* instruction to perform the interrupt return.
POP {PC}, or POP {…, PC}	Very often the value of LR is pushed to the stack after entering the exception handler. We can use the POP instruction, either a single POP or multiple POPs, to put the EXC_RETURN value to the program counter. This will cause the processor to perform the interrupt return.
LDR, or LDM	It is possible to produce an interrupt return using the LDR instruction with PC as the destination register.

Registers



- Handler mode and Thread mode
  - Handler mode always use MSP
  - Thread Mode uses either PSP or MSP
    - Control[1] = 0, SP = MSP (default)
    - Control[1] = 1, SP = PSP

MSP: Main Stack Pointer
PSP: Process Stack Pointer

Which stack to use when exiting an interrupt?

- EXC_RETURN indicates processor mode and stack type to be activated when exiting an interrupt
- EXC_RETURN is generated dynamically according to processor mode and stack type on that point

No FP extension:

EXC_RETURN	Return Mode	Return Stack
`0xFFFFFFF1`	Handler	SP = MSP
`0xFFFFFFF9`	Thread	SP = MSP
`0xFFFFFFFD`	Thread	SP = PSP

With FP extension:

EXC_RETURN	Return Mode	Return Stack
`0xFFFFFFE1`	Handler	SP = MSP
`0xFFFFFFE9`	Thread	SP = MSP
`0xFFFFFFED`	Thread	SP = PSP

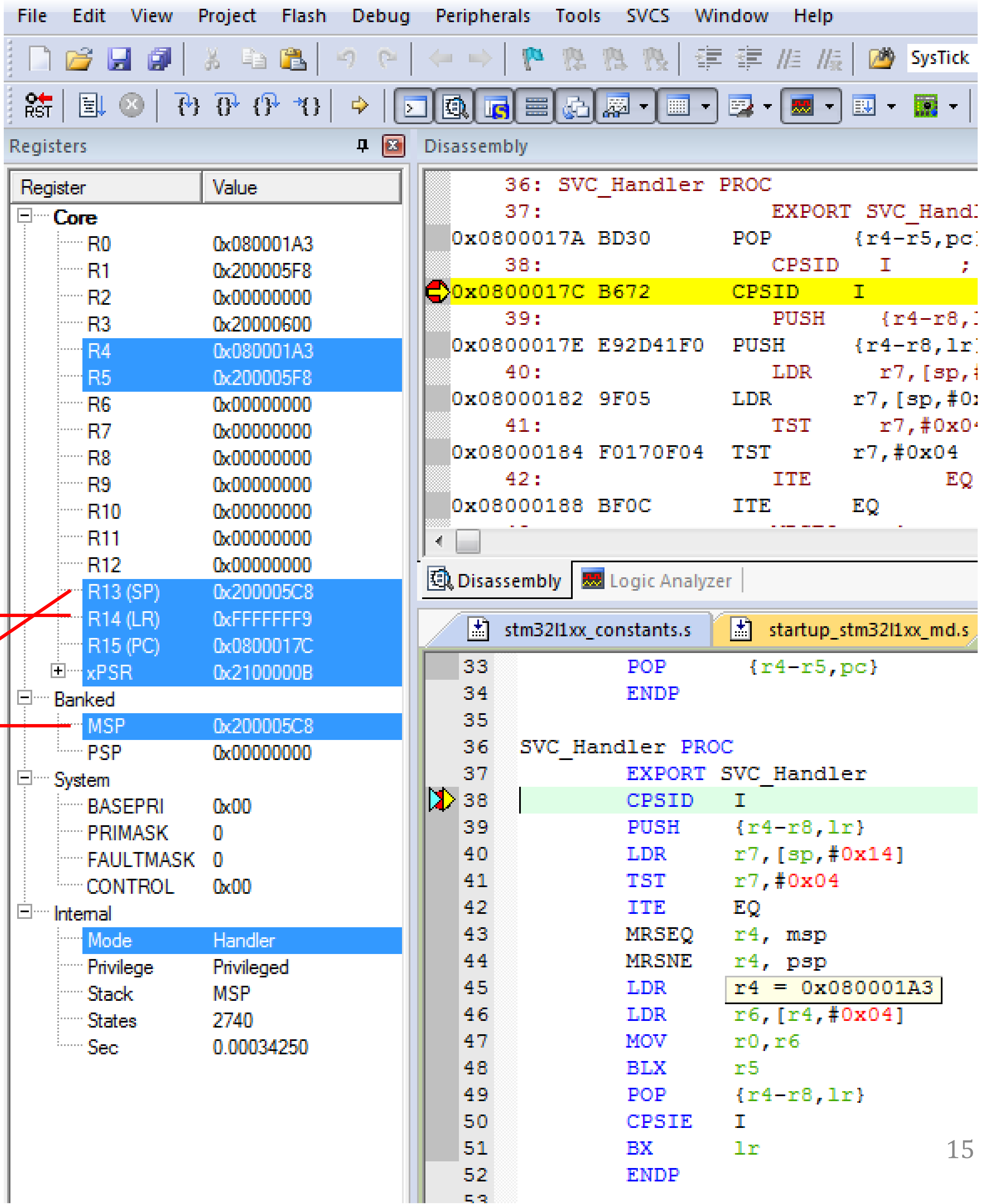
Register values in an interrupt service routine

An interrupt from the user-mode with the MSP

LR = 0xFFFFFFF9

SP = MSP

ISRs always run on the Handler mode with the MSP



Register	Value
Core	
R0	0x080001A3
R1	0x200005F8
R2	0x00000000
R3	0x20000600
R4	0x080001A3
R5	0x200005F8
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200005C8
R14 (LR)	0xFFFFFFF9
R15 (PC)	0x0800017C
xPSR	0x2100000B
Banked	
MSP	0x200005C8
PSP	0x00000000
System	
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00
Internal	
Mode	Handler
Privilege	Privileged
Stack	MSP
States	2740
Sec	0.00034250

```
    36: SVC_Handler PROC
    37:                  EXPORT SVC_Handl
0x0800017A BD30      POP      {r4-r5,pc}
    38:                  CPSID   I      ;
0x0800017C B672      CPSID     I
    39:                  PUSH    {r4-r8,l
0x0800017E E92D41F0  PUSH     {r4-r8,lr}
    40:                  LDR     r7,[sp,#
0x08000182 9F05      LDR      r7,[sp,#0x
    41:                  TST     r7,#0x0
0x08000184 F0170F04  TST      r7,#0x04
    42:                  ITE          EQ
0x08000188 BF0C      ITE      EQ
```

```
            POP     {r4-r5,pc}
            ENDP

SVC_Handler PROC
            EXPORT  SVC_Handler
            CPSID   I
            PUSH    {r4-r8,lr}
            LDR     r7,[sp,#0x14]
            TST     r7,#0x04
            ITE     EQ
            MRSEQ   r4, msp
            MRSNE   r4, psp
            LDR     r4 = 0x080001A3
            LDR     r6,[r4,#0x04]
            MOV     r0,r6
            BLX     r5
            POP     {r4-r8,lr}
            CPSIE   I
            BX      lr
            ENDP
```

Interrupt: Stacking & Unstacking

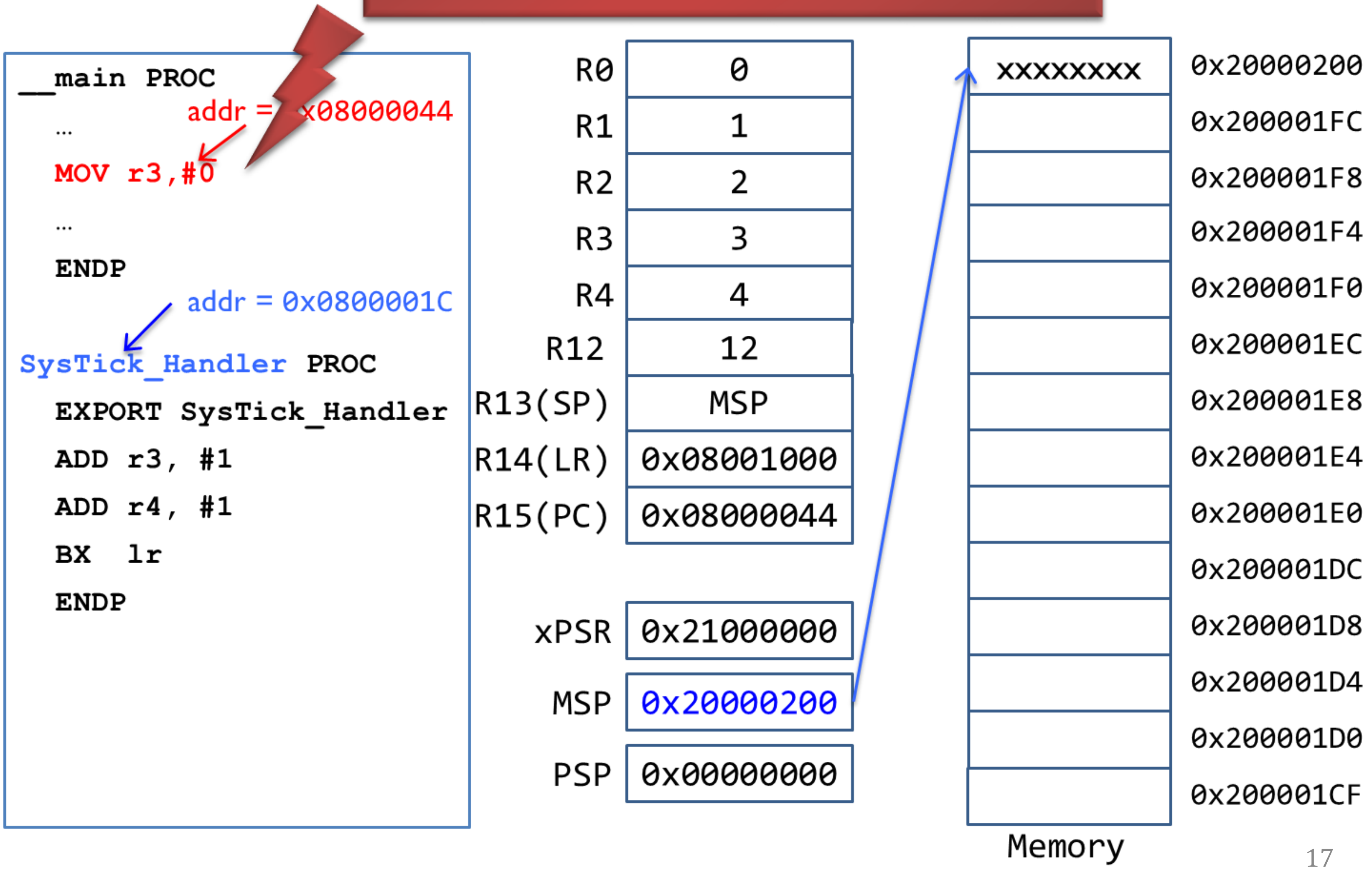
```
__main PROC
                addr = 0x08000044
  …
  MOV r3,#0
  …
  ENDP
                addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r3, #1
  ADD r4, #1
  BX  lr
  ENDP
```

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08001000
R15(PC)	0x08000044
xPSR	0x21000000
MSP	0x20000200
PSP	0x00000000

xxxxxxxx	0x20000200
	0x200001FC
	0x200001F8
	0x200001F4
	0x200001F0
	0x200001EC
	0x200001E8
	0x200001E4
	0x200001E0
	0x200001DC
	0x200001D8
	0x200001D4
	0x200001D0
	0x200001CF

Interrupt: Stacking & Unstacking

Suppose SysTick interrupt occurs when PC = 0x08000044



```
__main PROC
                  addr = 0x08000044
   ...
   MOV r3,#0
   ...
   ENDP
                  addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

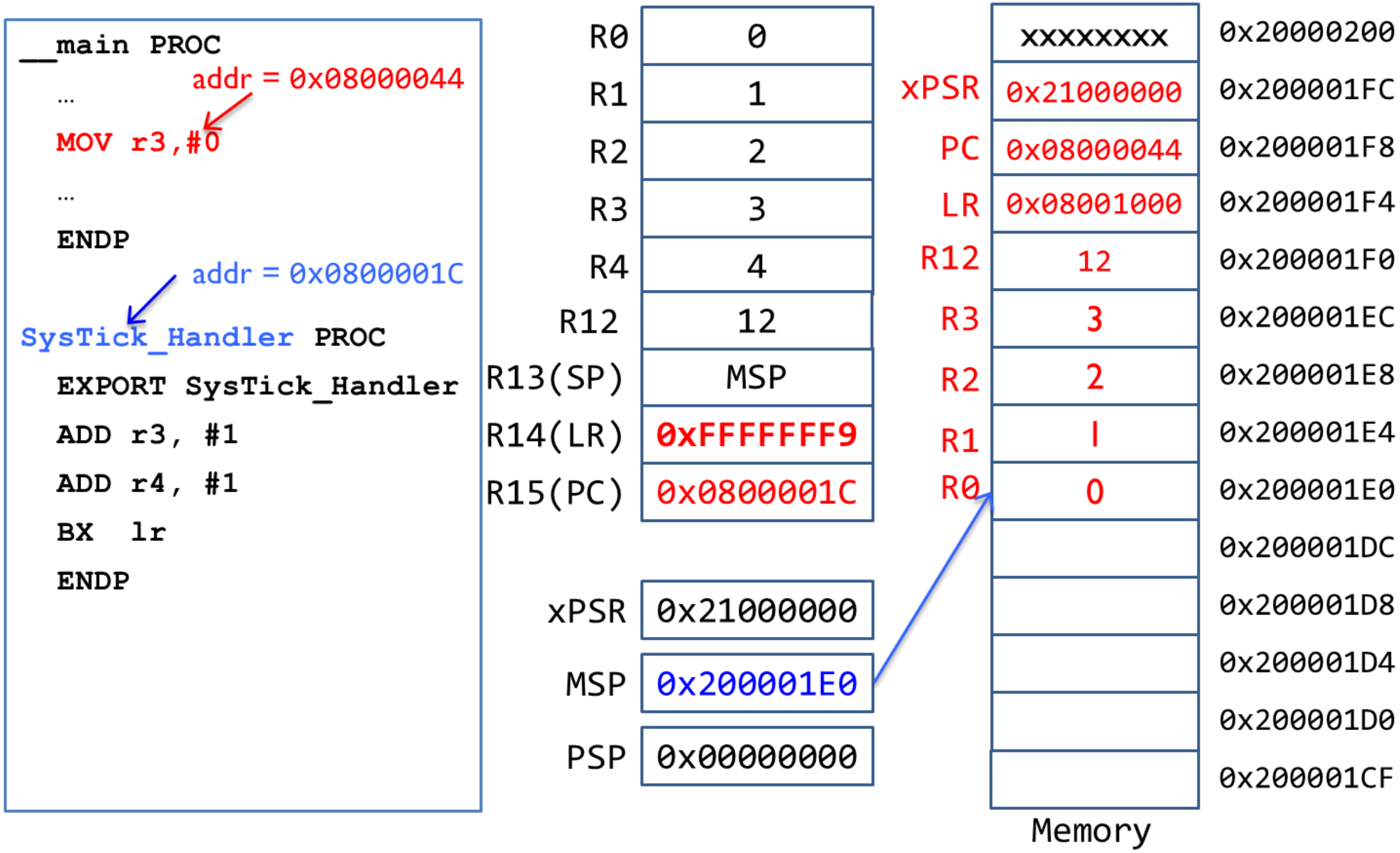
R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08001000
R15(PC)	0x08000044
xPSR	0x21000000
MSP	0x20000200
PSP	0x00000000

xxxxxxxx	0x20000200
	0x200001FC
	0x200001F8
	0x200001F4
	0x200001F0
	0x200001EC
	0x200001E8
	0x200001E4
	0x200001E0
	0x200001DC
	0x200001D8
	0x200001D4
	0x200001D0
	0x200001CF

Memory

Interrupt: Stacking & Unstacking

STACKING



```
__main PROC
                  addr = 0x08000044
   ...
   MOV r3,#0
   ...
   ENDP
                  addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x0800001C
xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking

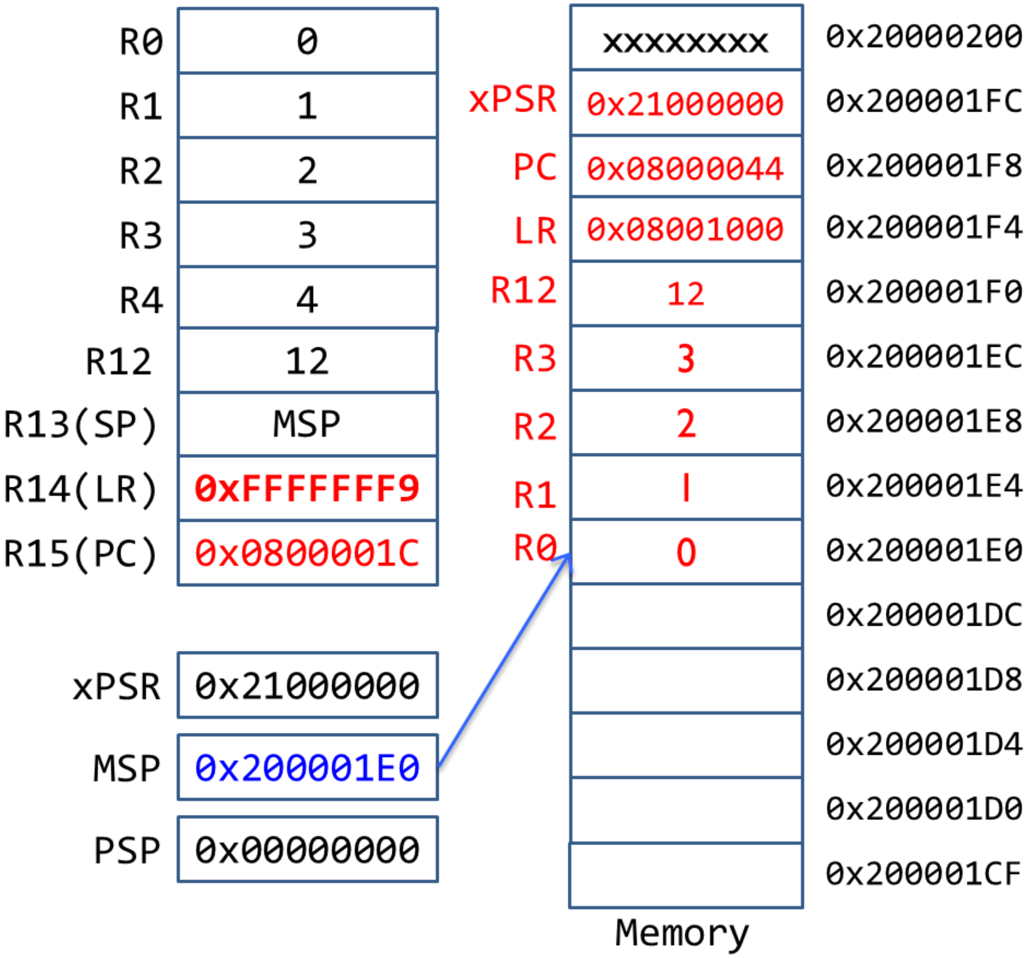
STACKING

```
__main PROC
              addr = 0x08000044
   ...
   MOV r3,#0

   ...
   ENDP
              addr = 0x0800001C

SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

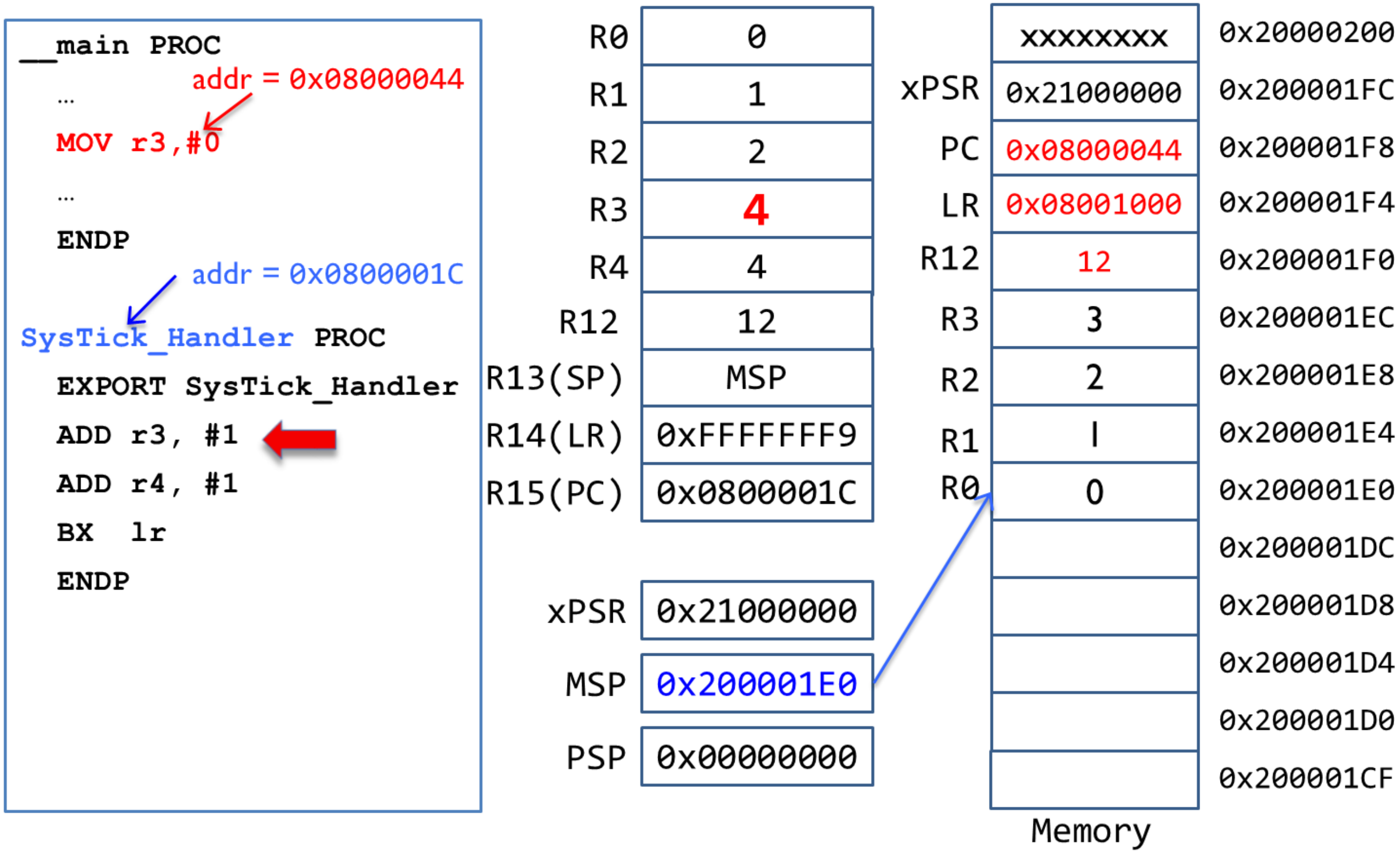
LR = 0xFFFFFFF9 to indicate MSP is used.



R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x0800001C
xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	Memory	
	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Interrupt: Stacking & Unstacking



```
__main PROC
                addr = 0x08000044
   ...
   MOV r3,#0

   ...
   ENDP
                addr = 0x0800001C

SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

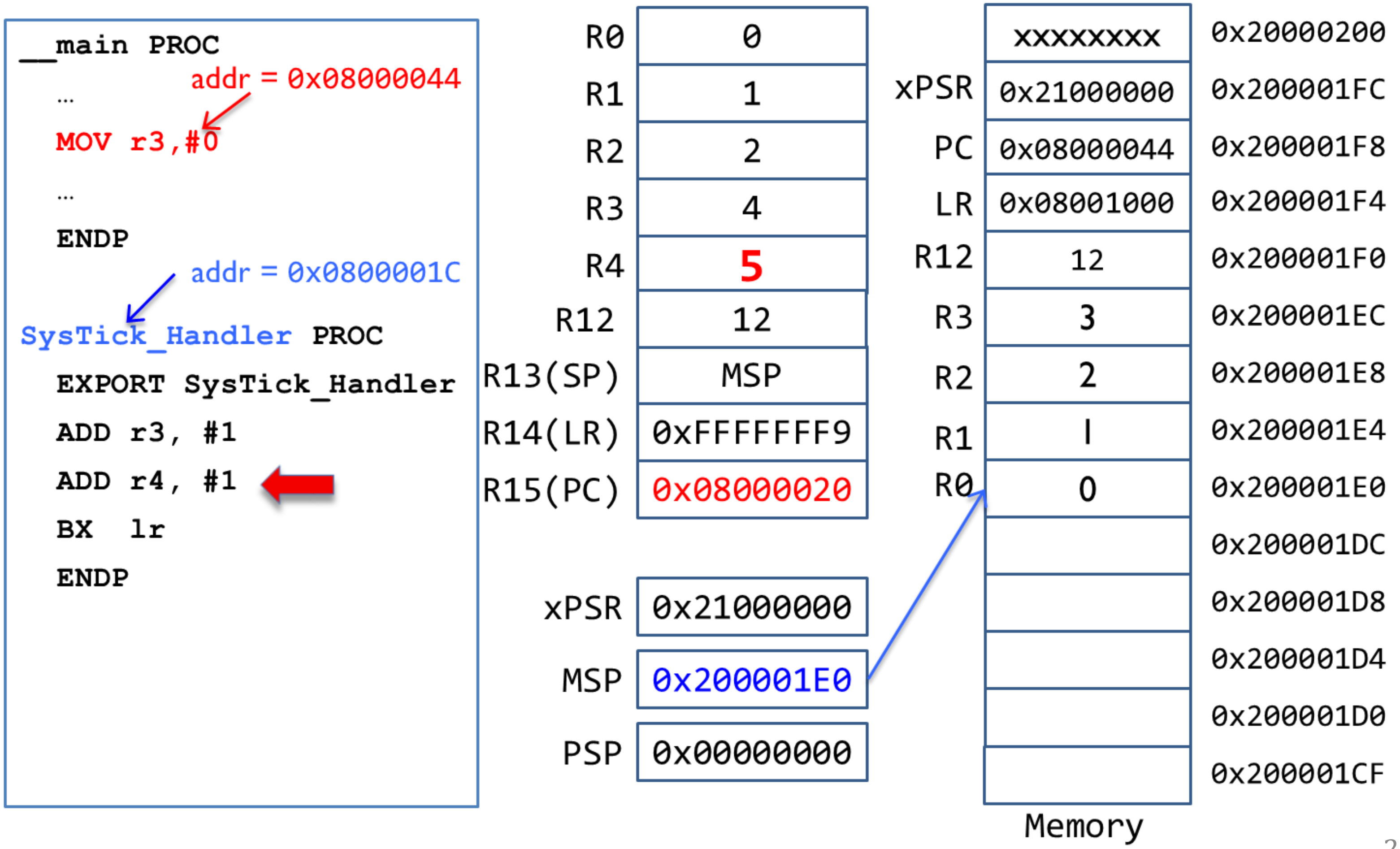
R0	0
R1	1
R2	2
R3	4
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x0800001C

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking



```
__main PROC
              addr = 0x08000044
  ...
  MOV r3,#0
  ...
  ENDP
              addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r3, #1
  ADD r4, #1
  BX  lr
  ENDP
```

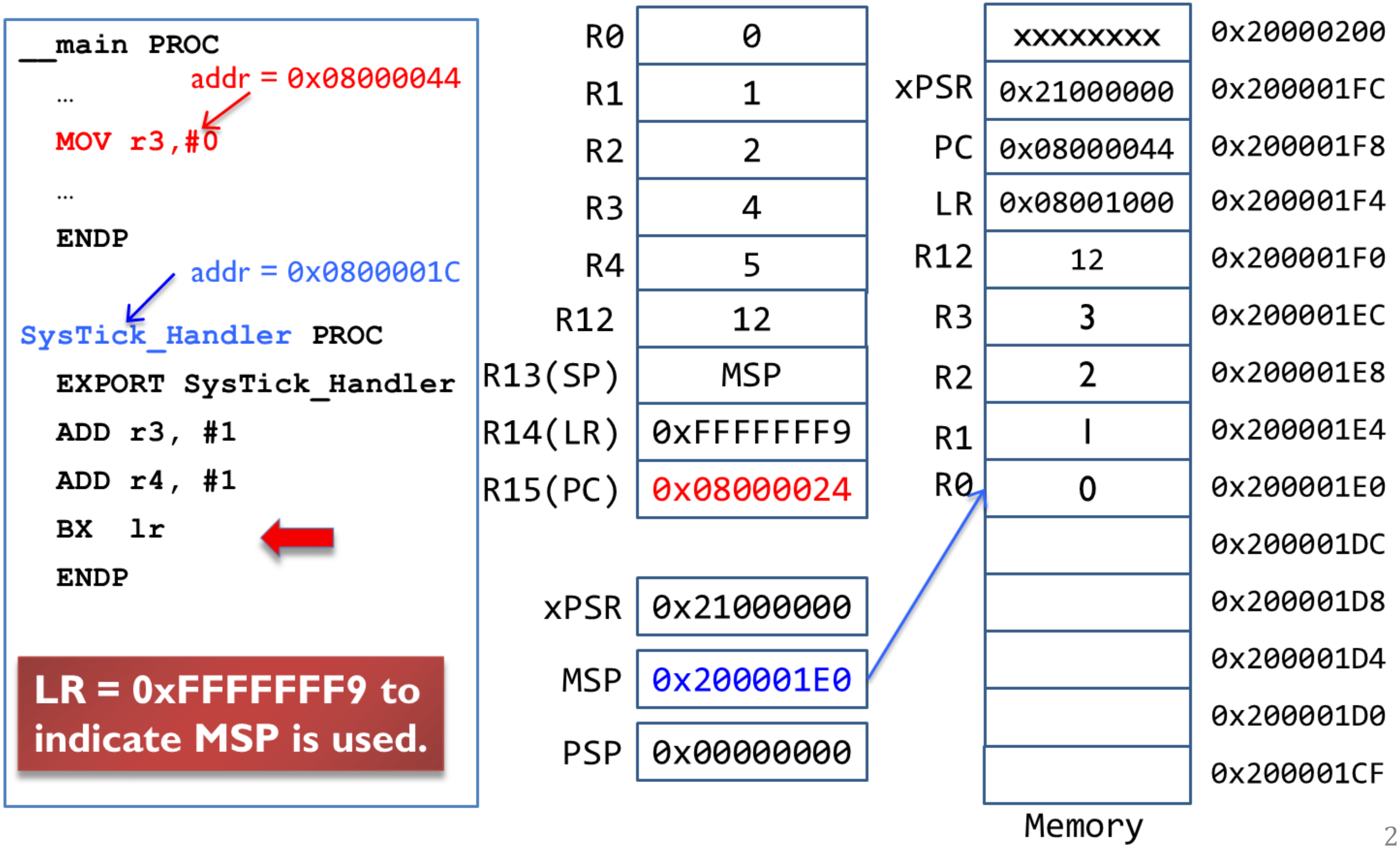
R0	0
R1	1
R2	2
R3	4
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x08000020

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	1	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking



```
__main PROC
            addr = 0x08000044
   ...
   MOV r3,#0
   ...
   ENDP
            addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

LR = 0xFFFFFFF9 to indicate MSP is used.

R0	0
R1	1
R2	2
R3	4
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x08000024

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

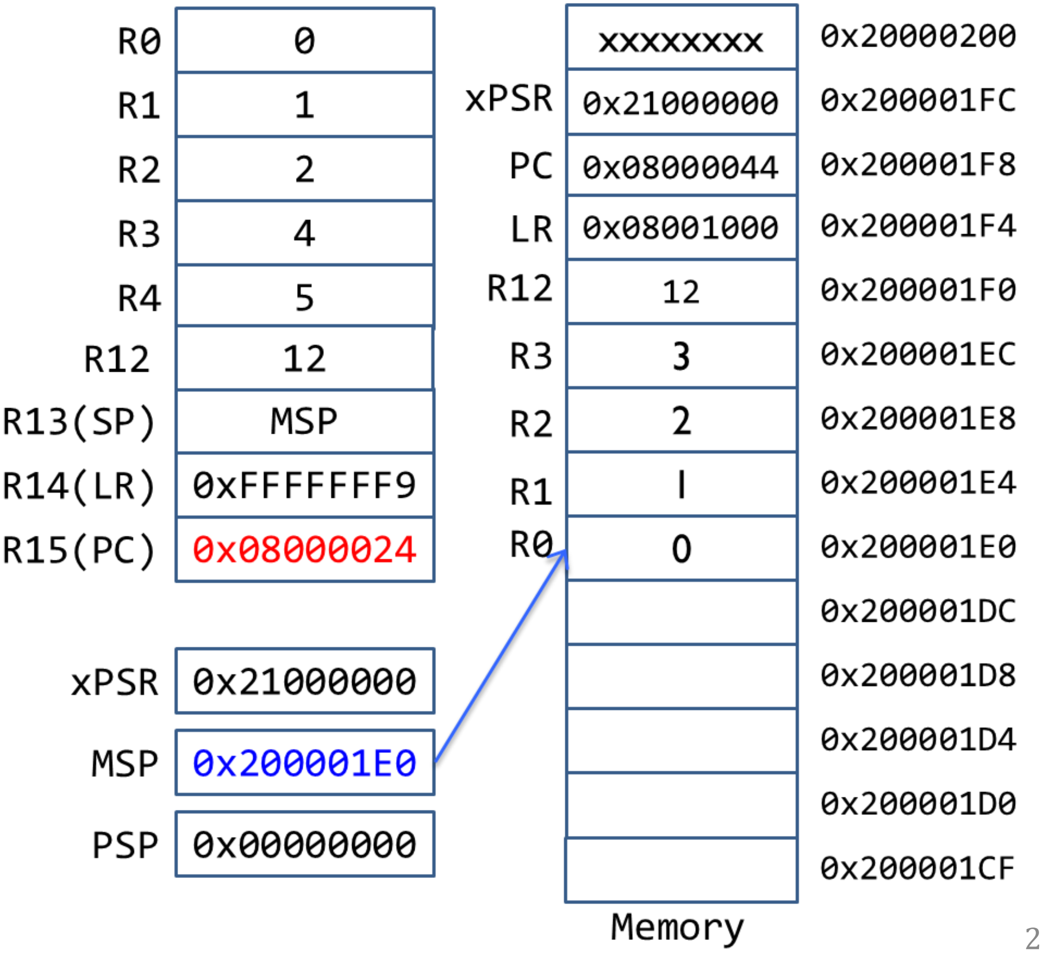
	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Interrupt: Stacking & Unstacking

UNSTACKING

```
__main PROC
                addr = 0x08000044
  ...
  MOV r3,#0
  ...
  ENDP
                addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r3, #1
  ADD r4, #1
  BX  lr
  ENDP
```

LR = 0xFFFFFFF9 to indicate MSP is used.



R0	0
R1	1
R2	2
R3	4
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x08000024

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
LR	0x08001000	0x200001F4
R12	12	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

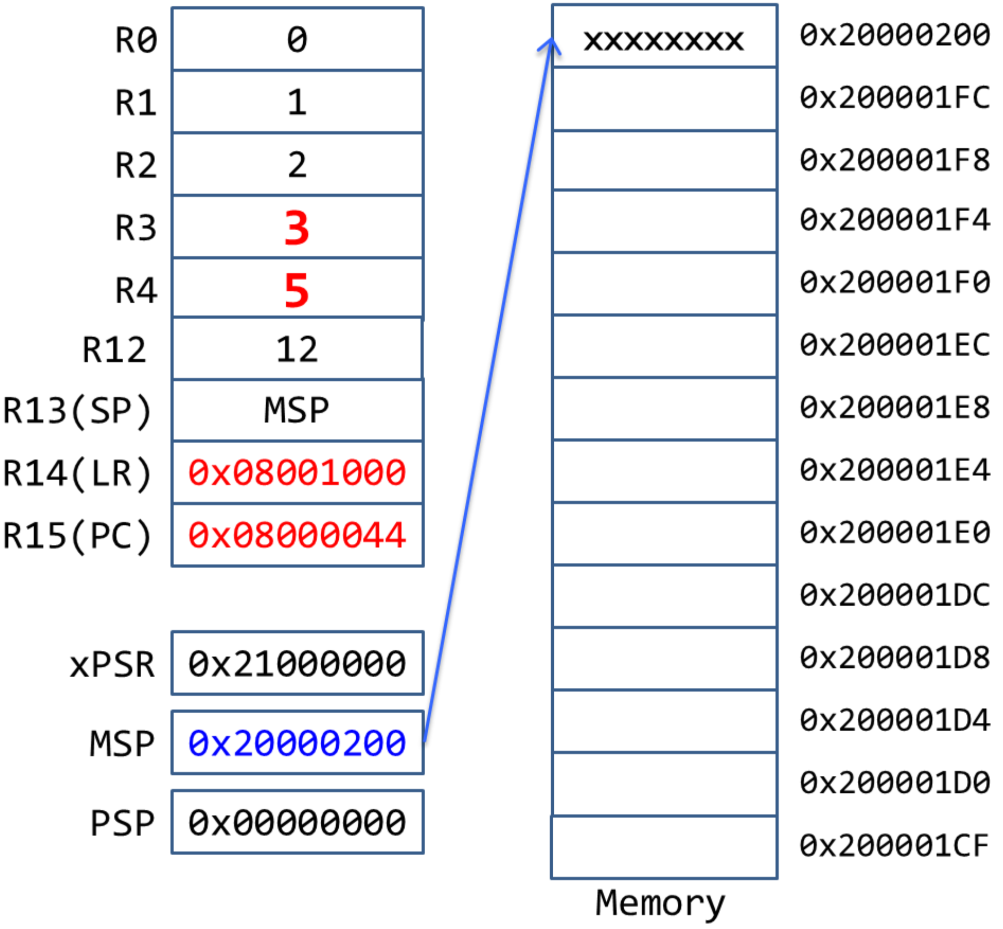
Memory

Interrupt: Stacking & Unstacking

UNSTACKING

```
__main PROC
            addr = 0x08000044
  ...
  MOV r3,#0
  ...
  ENDP
            addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r3, #1
  ADD r4, #1
  BX  lr
  ENDP
```

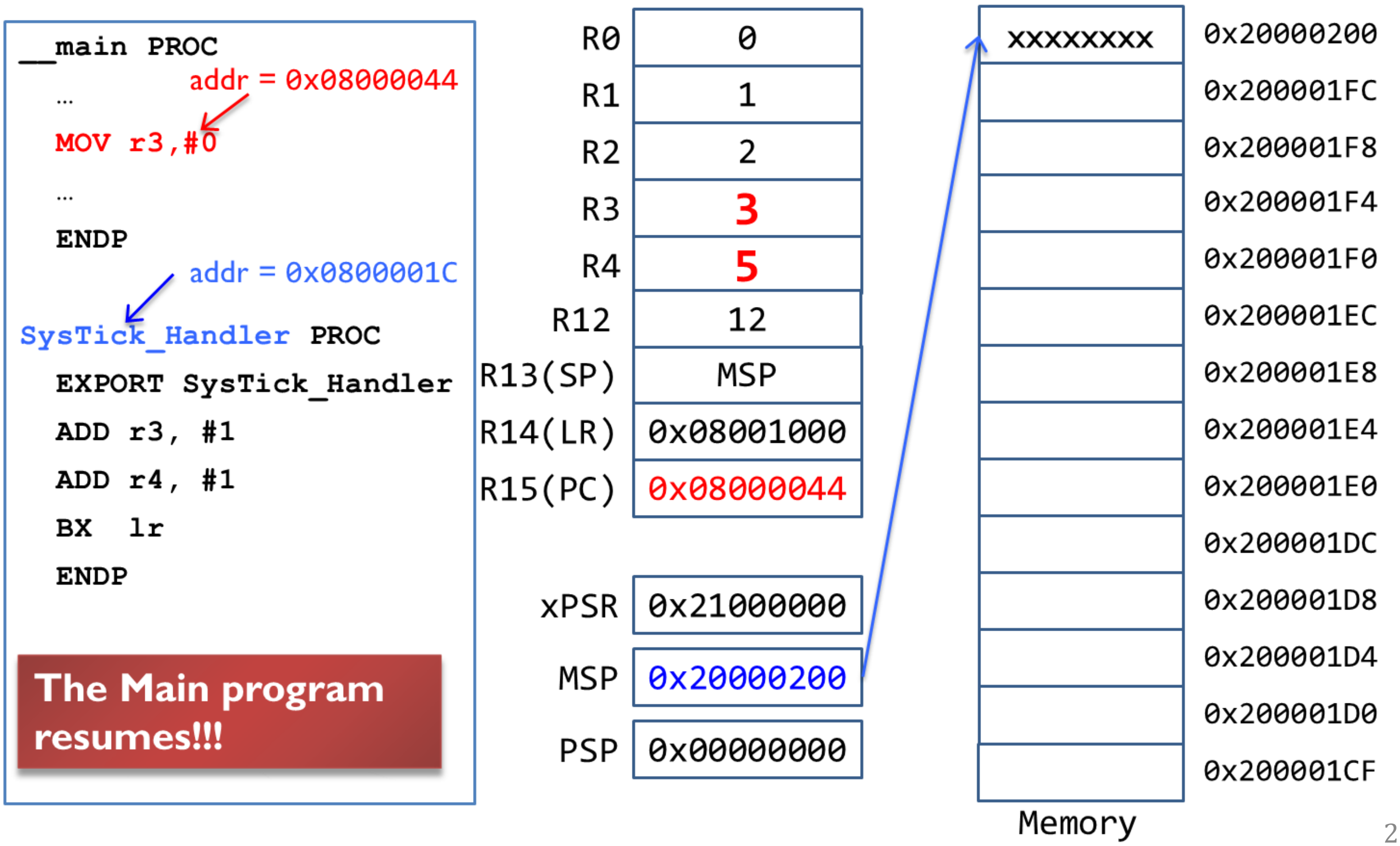
Note the new value of R3 is lost!!!



Register	Value
R0	0
R1	1
R2	2
R3	3
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0x08001000
R15(PC)	0x08000044
xPSR	0x21000000
MSP	0x20000200
PSP	0x00000000

Memory	Address
xxxxxxxx	0x20000200
	0x200001FC
	0x200001F8
	0x200001F4
	0x200001F0
	0x200001EC
	0x200001E8
	0x200001E4
	0x200001E0
	0x200001DC
	0x200001D8
	0x200001D4
	0x200001D0
	0x200001CF

Interrupt: Stacking & Unstacking



```
__main PROC
   ...
   MOV r3,#0
   ...
   ENDP

SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r3, #1
   ADD r4, #1
   BX  lr
   ENDP
```

addr = 0x08000044

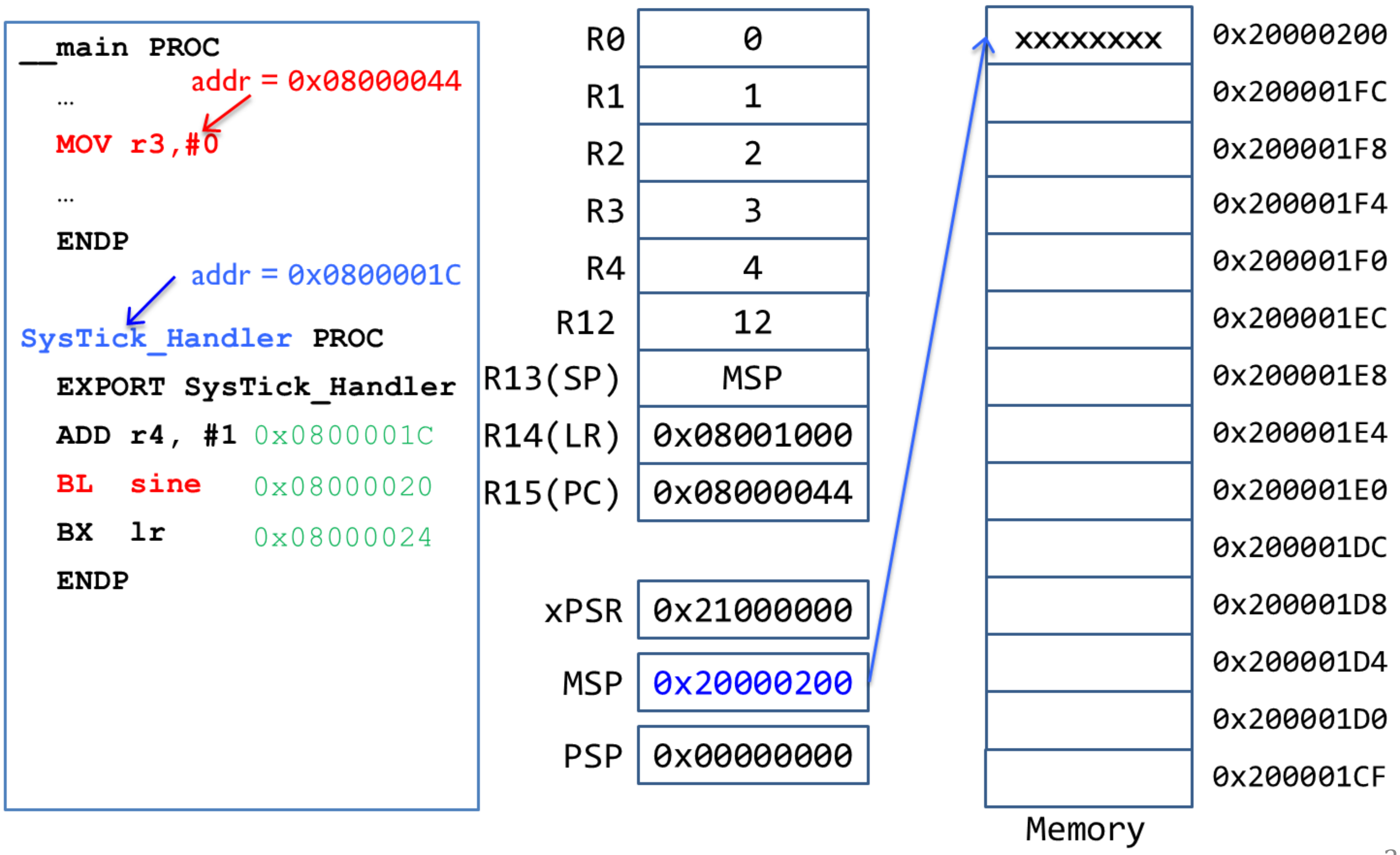
addr = 0x0800001C

The Main program resumes!!!

R0	0
R1	1
R2	2
R3	3
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0x08001000
R15(PC)	0x08000044
xPSR	0x21000000
MSP	0x20000200
PSP	0x00000000

Memory	
xxxxxxxx	0x20000200
	0x200001FC
	0x200001F8
	0x200001F4
	0x200001F0
	0x200001EC
	0x200001E8
	0x200001E4
	0x200001E0
	0x200001DC
	0x200001D8
	0x200001D4
	0x200001D0
	0x200001CF

Interrupt: Stacking & Unstacking



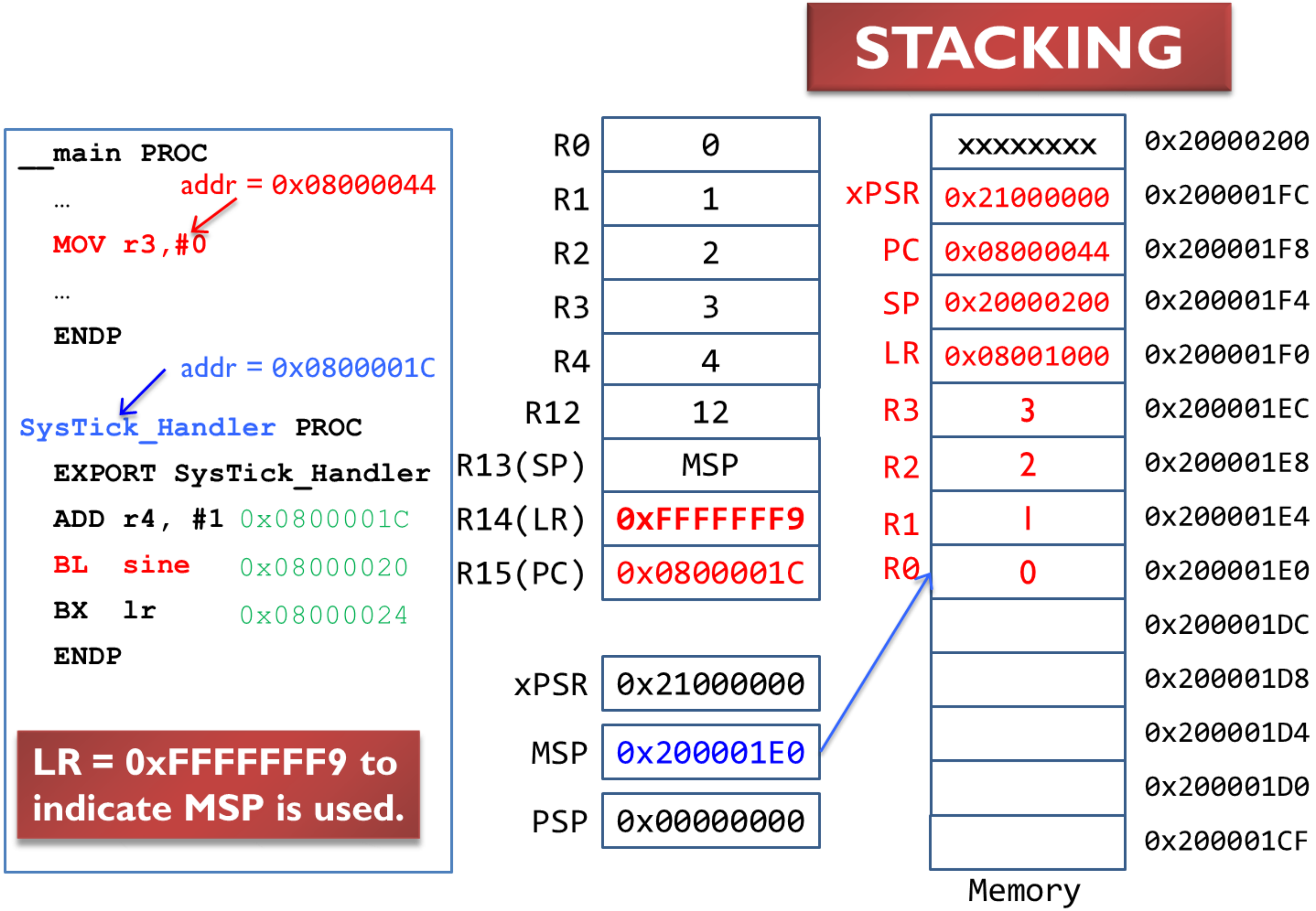
```
__main PROC
                 addr = 0x08000044
   …
   MOV r3,#0
   …
   ENDP
                 addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r4, #1   0x0800001C
   BL  sine     0x08000020
   BX  lr       0x08000024
   ENDP
```

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08001000
R15(PC)	0x08000044
xPSR	0x21000000
MSP	0x20000200
PSP	0x00000000

xxxxxxxx	0x20000200
	0x200001FC
	0x200001F8
	0x200001F4
	0x200001F0
	0x200001EC
	0x200001E8
	0x200001E4
	0x200001E0
	0x200001DC
	0x200001D8
	0x200001D4
	0x200001D0
	0x200001CF

Memory

Interrupt: Stacking & Unstacking



STACKING

```
__main PROC
                  addr = 0x08000044
  ...
  MOV r3,#0
  ...
  ENDP
                  addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r4, #1    0x0800001C
  BL  sine      0x08000020
  BX  lr        0x08000024
  ENDP
```

LR = 0xFFFFFFF9 to indicate MSP is used.

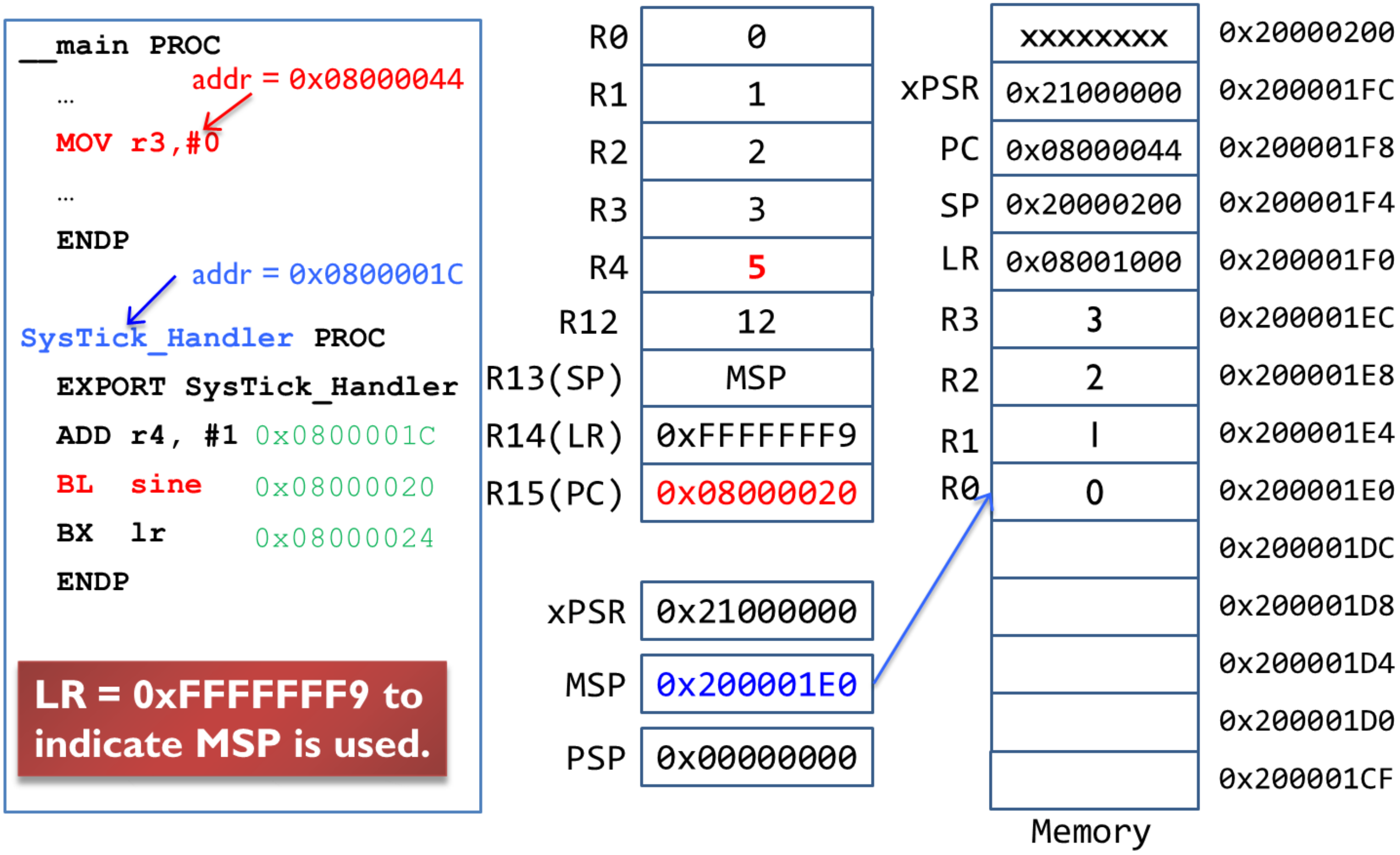
R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x0800001C

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	1	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking



```
__main  PROC
                addr = 0x08000044
   ...
   MOV  r3,#0
   ...
   ENDP
                addr = 0x0800001C
SysTick_Handler  PROC
   EXPORT  SysTick_Handler
   ADD  r4, #1   0x0800001C
   BL   sine     0x08000020
   BX   lr       0x08000024
   ENDP
```

LR = 0xFFFFFFF9 to indicate MSP is used.

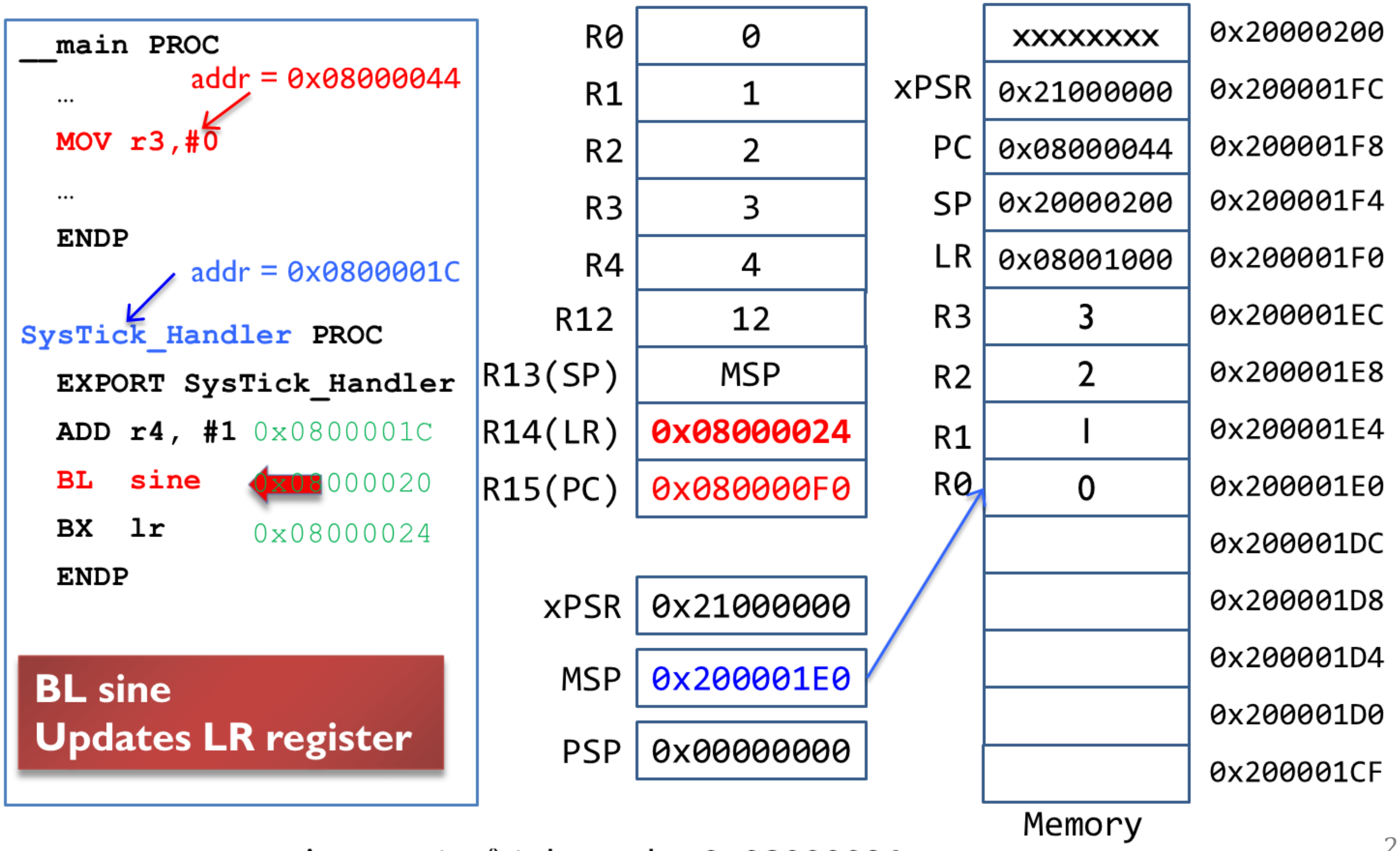
R0	0
R1	1
R2	2
R3	3
R4	5
R12	12
R13(SP)	MSP
R14(LR)	0xFFFFFFF9
R15(PC)	0x08000020

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking



```
__main PROC
                addr = 0x08000044
   ...
   MOV r3,#0

   ...
   ENDP
                addr = 0x0800001C

SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r4, #1  0x0800001C
   BL  sine     0x08000020
   BX  lr       0x08000024
   ENDP
```

BL sine
Updates LR register

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08000024
R15(PC)	0x080000F0

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x08000044	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Assume sine() is located at 0x08000024.

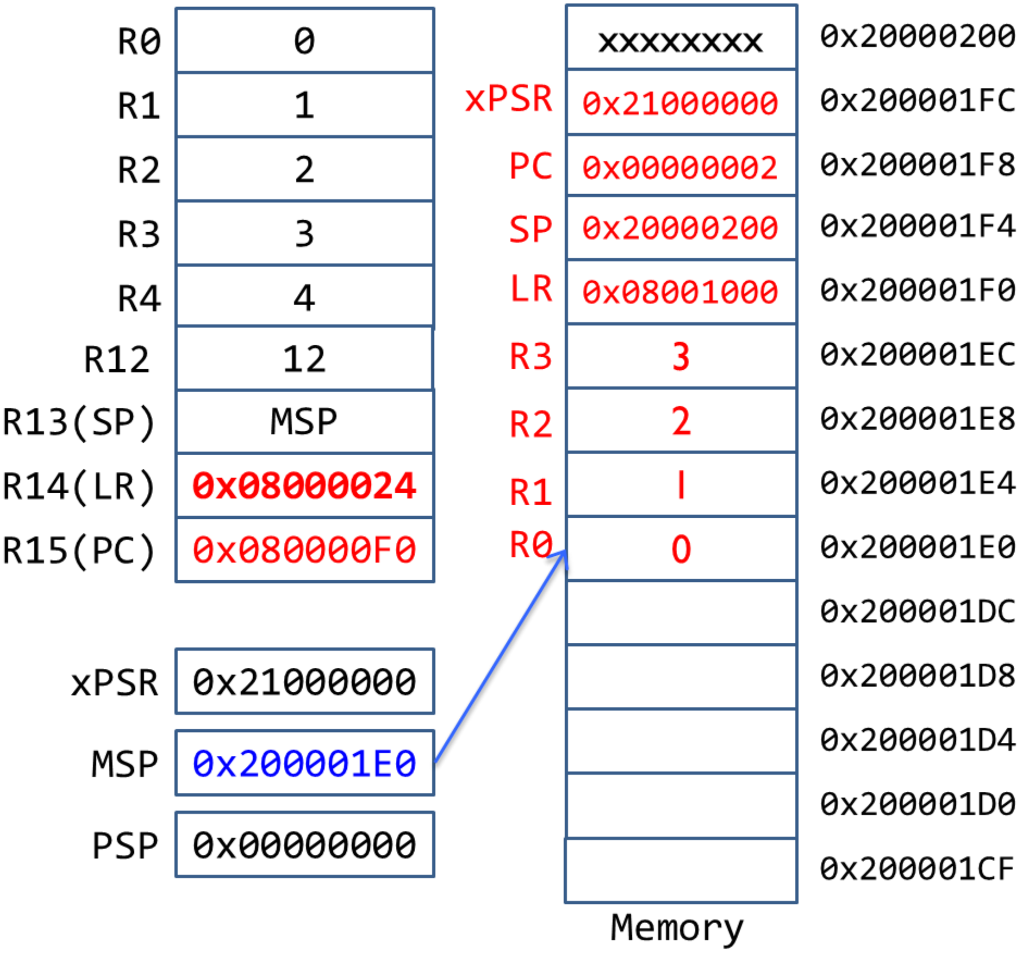
Interrupt: Stacking & Unstacking

```
__main PROC
                  addr = 0x08000044
   ...
   MOV r3,#0

   ...
   ENDP
                  addr = 0x0800001C

SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r4, #1     0x0800001C
   BL  sine       0x08000020
   BX  lr         0x08000024
   ENDP
```

BL sine
Updates LR register



R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08000024
R15(PC)	0x080000F0

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	Memory	
	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x00000002	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	1	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Interrupt: Stacking & Unstacking

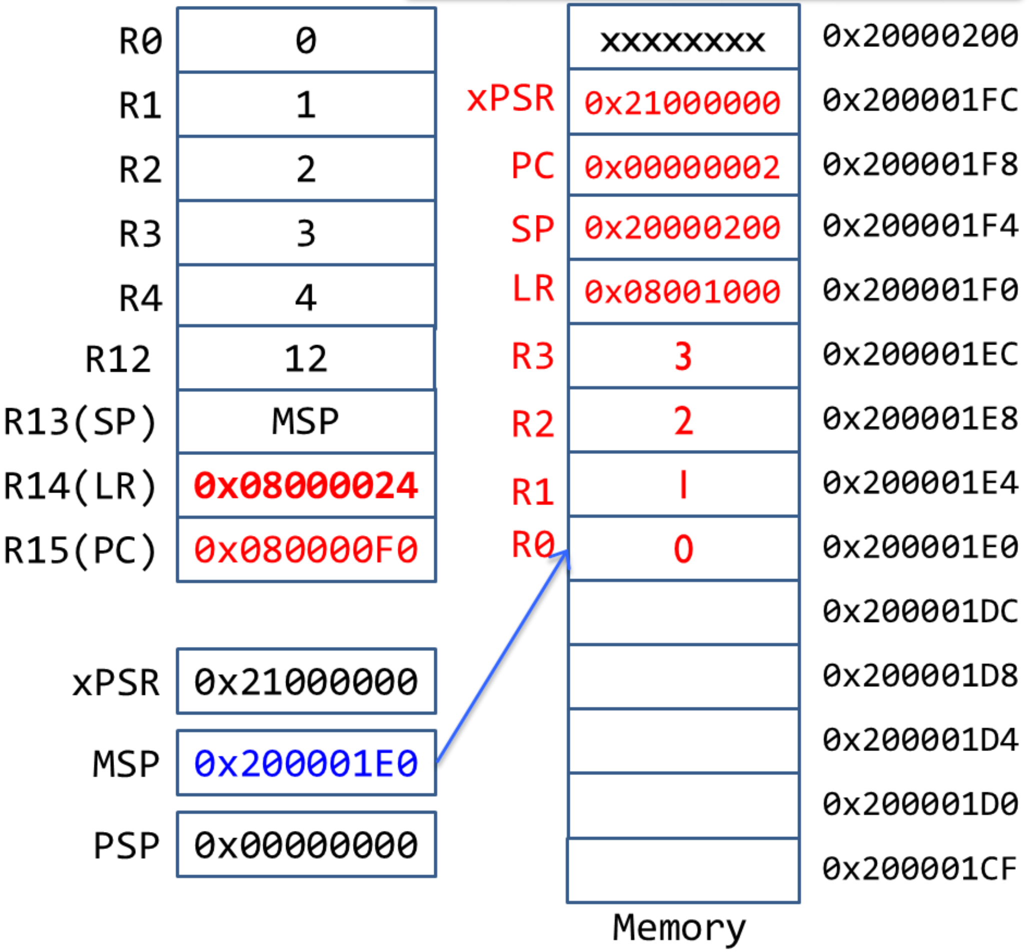
UNSTACKING won't occur!

```
__main PROC
                addr = 0x08000044
  …
  MOV r3,#0

  …
  ENDP
                addr = 0x0800001C

SysTick_Handler PROC
  EXPORT SysTick_Handler
  ADD r4, #1      0x0800001C
  BL  sine        0x08000020
  BX  lr          0x08000024
  ENDP
```

BL sine
Updates LR register



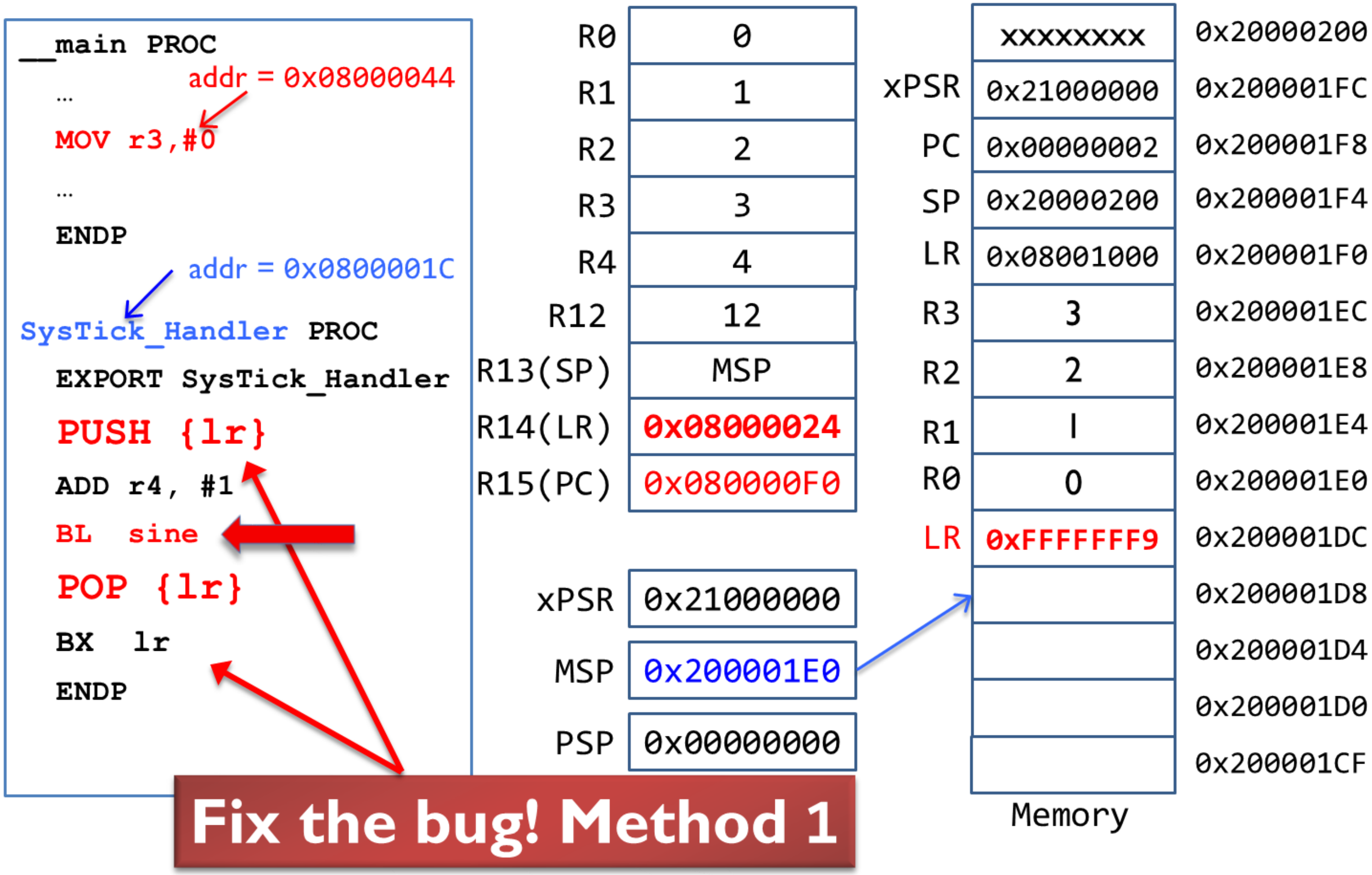
R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08000024
R15(PC)	0x080000F0

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x00000002	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Interrupt: Stacking & Unstacking



```
__main PROC
            addr = 0x08000044
  ...
  MOV r3,#0
  ...
  ENDP
            addr = 0x0800001C
SysTick_Handler PROC
  EXPORT SysTick_Handler
  PUSH {lr}
  ADD r4, #1
  BL  sine
  POP {lr}
  BX  lr
  ENDP
```

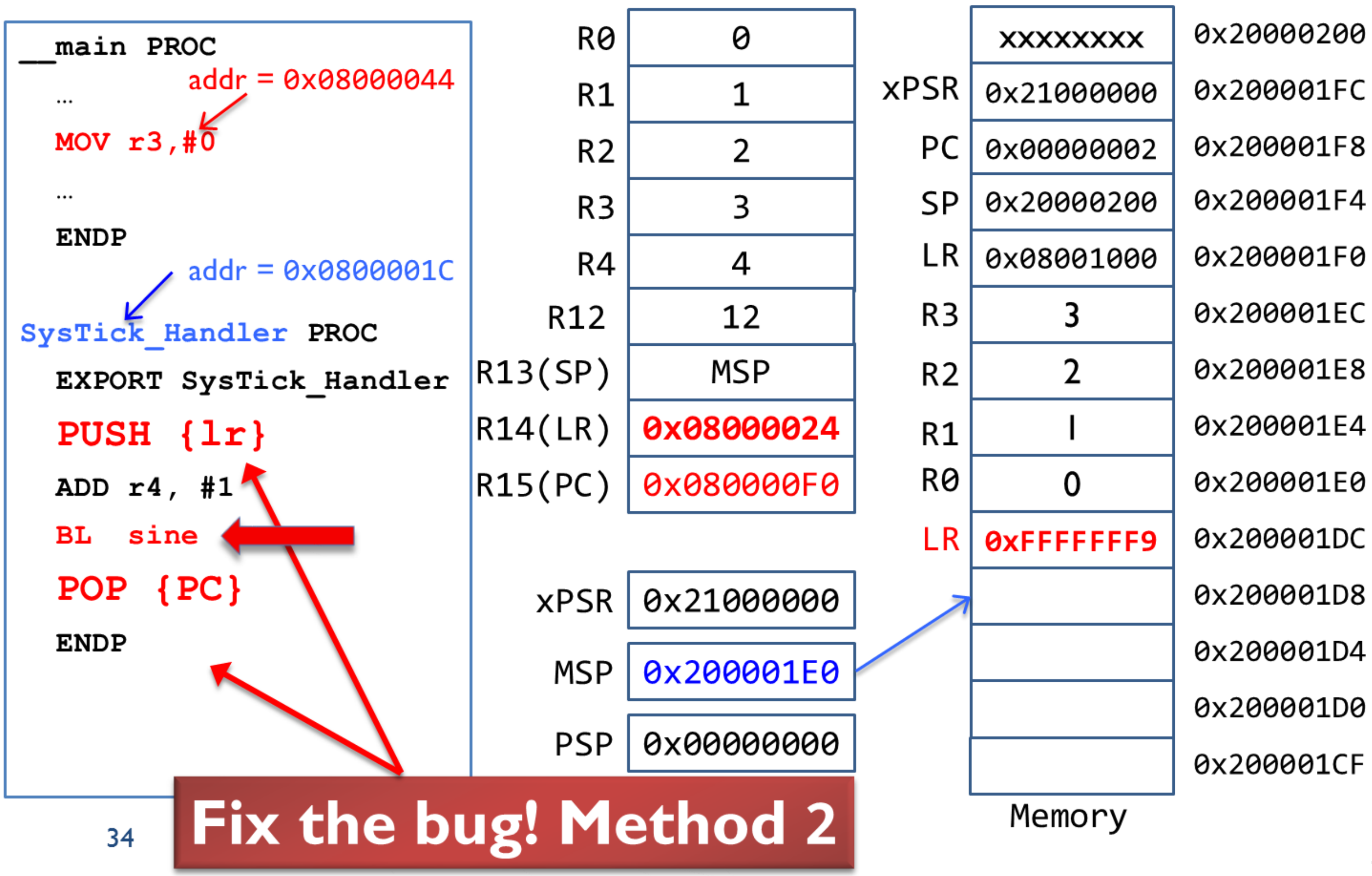
R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08000024
R15(PC)	0x080000F0
xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x00000002	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
LR	0xFFFFFFF9	0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

Fix the bug! Method 1

Interrupt: Stacking & Unstacking



```
__main PROC
            addr = 0x08000044
   …
   MOV r3,#0
   …
   ENDP
            addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   PUSH {lr}
   ADD r4, #1
   BL  sine
   POP {PC}
   ENDP
```

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	0x08000024
R15(PC)	0x080000F0

xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

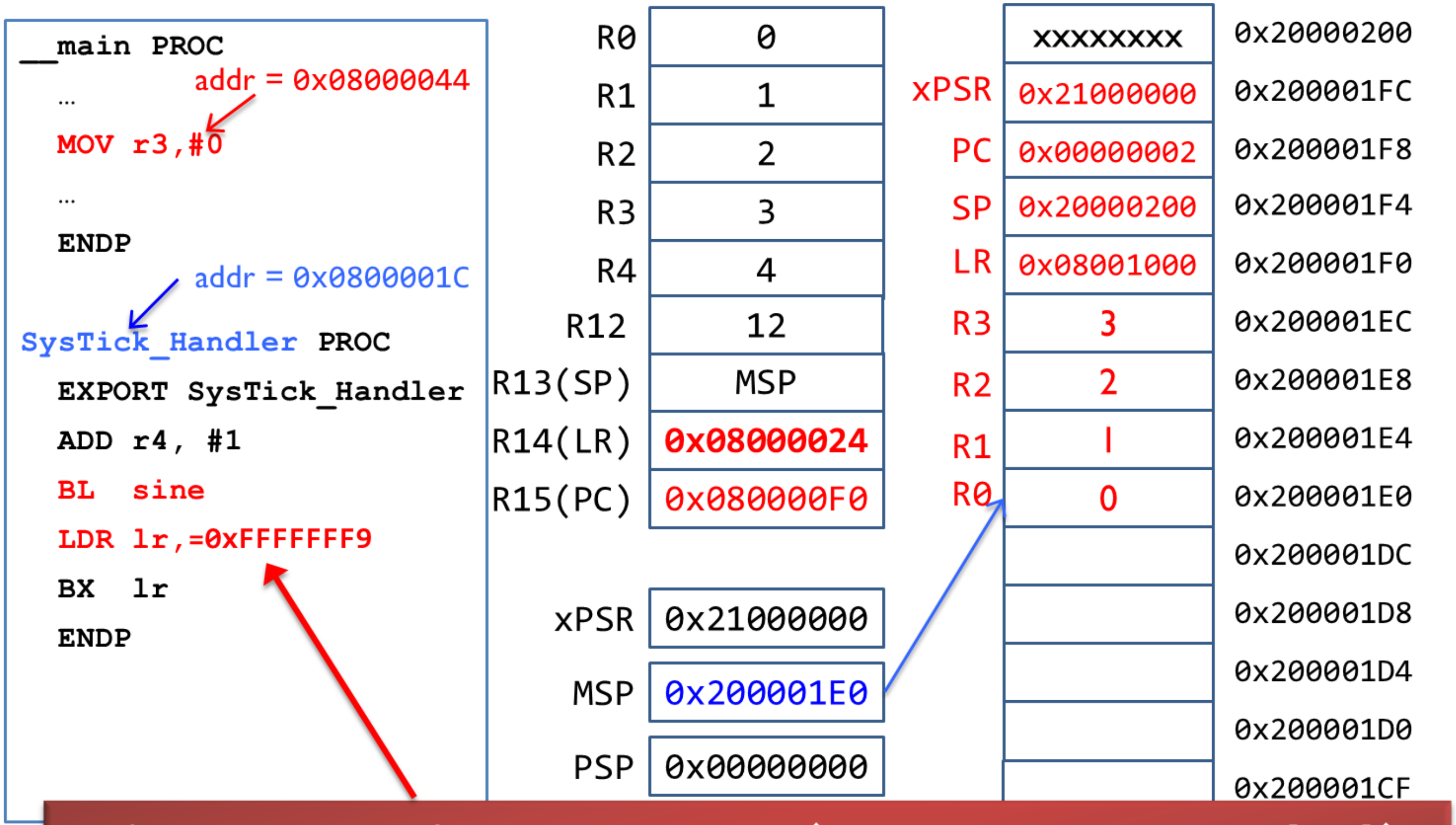
	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x00000002	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
LR	0xFFFFFFF9	0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Memory

34

Fix the bug! Method 2

Interrupt: Stacking & Unstacking



```
__main PROC
            addr = 0x08000044
   ...
   MOV r3,#0
   ...
   ENDP
            addr = 0x0800001C
SysTick_Handler PROC
   EXPORT SysTick_Handler
   ADD r4, #1
   BL  sine
   LDR lr,=0xFFFFFFF9
   BX  lr
   ENDP
```

R0	0
R1	1
R2	2
R3	3
R4	4
R12	12
R13(SP)	MSP
R14(LR)	**0x08000024**
R15(PC)	0x080000F0

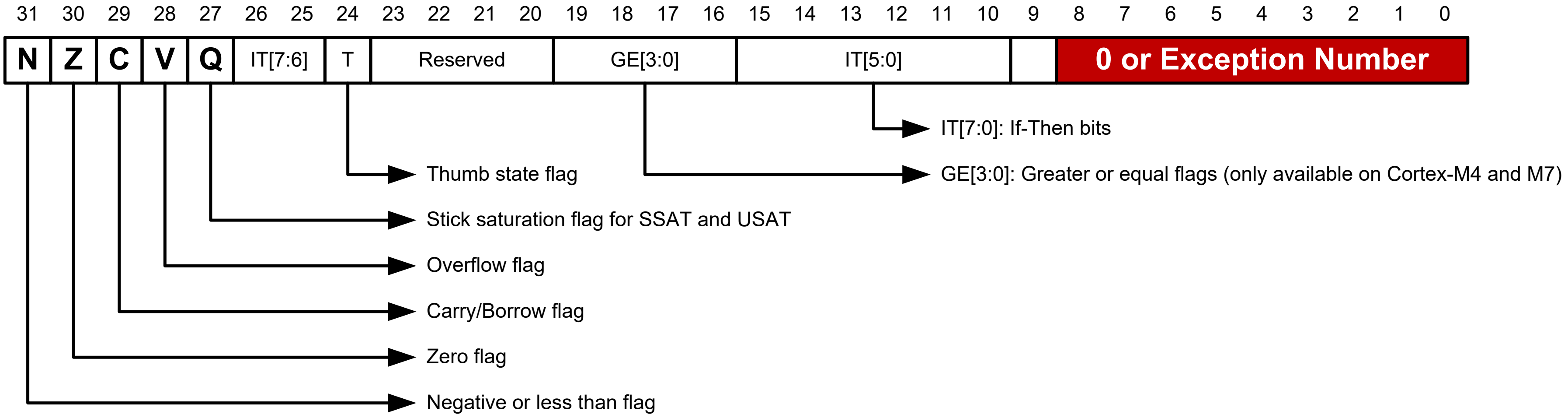
xPSR	0x21000000
MSP	0x200001E0
PSP	0x00000000

	xxxxxxxx	0x20000200
xPSR	0x21000000	0x200001FC
PC	0x00000002	0x200001F8
SP	0x20000200	0x200001F4
LR	0x08001000	0x200001F0
R3	3	0x200001EC
R2	2	0x200001E8
R1	l	0x200001E4
R0	0	0x200001E0
		0x200001DC
		0x200001D8
		0x200001D4
		0x200001D0
		0x200001CF

Fix the bug! Method 3 (not recommended)

Interrupt Number in PSR

- Valid exception numbers on ARMv7-M
    - 1 to 511



Enable an Interrupt/Exception

- Enable a system exception (0~15)
  - Some are always enabled (cannot be disabled)
  - No centralized registers for enabling/disabling
  - Each is controlled by its corresponding components, such as SysTick module

- Enable a peripheral interrupt (16~)
  - Centralized register arrays for enabling/disabling
  - NVIC's **`ISER`** 0~15 registers for enabling
    - Interrupt Set Enable Register
    - supports up to (32 * 16 = 512) interrupts
  - NVIC's **`ICER`** 0~15 registers for disabling
    - Interrupt Clear Enable Register
    - ICERs override ISERs

Enabling Peripheral Interrupts

Interrupt Set Enable Register 0 (ISER0)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	I2C1_EV	TIM4	TIM3	TIM2	TIM11	TIM10	TIM9	LCD	EXTI9_5	COMP	DAC	USB_LP	USB_HP	ADC1	DMA1_CH7	DMA1_CH6	DMA1_CH5	DMA1_CH4	DMA1_CH3	DMA1_CH2	DMA1_CH1	EXTI4	EXTI3	EXTI2	EXTI1	EXTI0	RCC	FLASH	RTC_WKUP	TAMPER_STAMP	PVD	WWDG

Interrupt Set Enable Register 1 (ISER1) *Address of ISER1 = Address of ISER0 + 4*

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number																				44	43	42	41	40	39	38	37	36	35	34	33	32
																				TIM7 ↑	TIM6	USB_FS_WKUP	RTC_Alarm	EXTI15_10	USART3	USART2	USART1	SPI2	SPI1	I2C2_ER	I2C2_EV	I2C1_ER

```
TIM7_IRQn = 44

NVIC->ISER[1] = 1 << 12;     // Enable Timer 7 interrupt
```

Disabling Peripheral Interrupts

Interrupt Clear Enable Register 0 (ICER0)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clear Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	I2C1_EV	TIM4	TIM3	TIM2	TIM11	TIM10	TIM9	LCD	EXTI9_5	COMP	DAC	USB_LP	USB_HP	ADC1	DMA1_CH7	DMA1_CH6	DMA1_CH5	DMA1_CH4	DMA1_CH3	DMA1_CH2	DMA1_CH1	EXTI4	EXTI3	EXTI2	EXTI1	EXTI0	RCC	FLASH	RTC_WKUP	TAMPER_STAMP	PVD	WWDG

Interrupt Clear Enable Register 1 (ICER1) ***Address of ICER1 = Address of ISER0 + 4***

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clear Enable Bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt Number																				44	43	42	41	40	39	38	37	36	35	34	33	32
																				TIM7	TIM6	USB_FS_WKUP	RTC_Alarm	EXTI15_10	USART3	USART2	USART1	SPI2	SPI1	I2C2_ER	I2C2_EV	I2C1_ER

```
TIM7_IRQn = 44

NVIC->ICER[1] = 1 << 12;      // Diable Timer 7 interrupt
```

Priority Management

Interrupt/Exception Priority

- Inverse Relationship:
  - Lower priority value means higher urgency.
    - Priority of Interrupt A = 5,
    - Priority of Interrupt B = 2,
    - B has a higher priority/urgency than A.

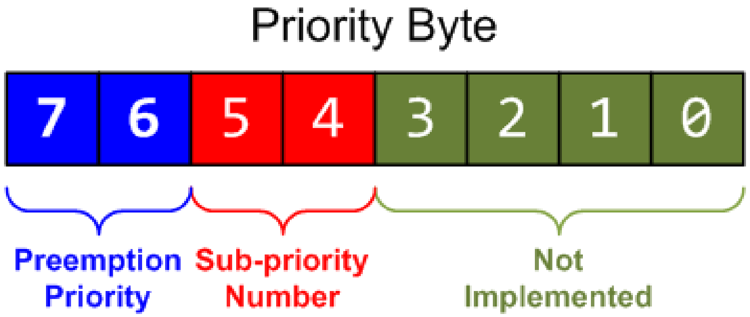
- Fixed priority for Reset, HardFault, and NMI.

Exception	**IRQn**	**Priority**
Reset	N/A	-3 (the highest)
Non-maskable Interrupt (NMI)	14	-2 (2nd highest)
Hard Fault	13	-1

- Adjustable for all the other interrupts and exceptions

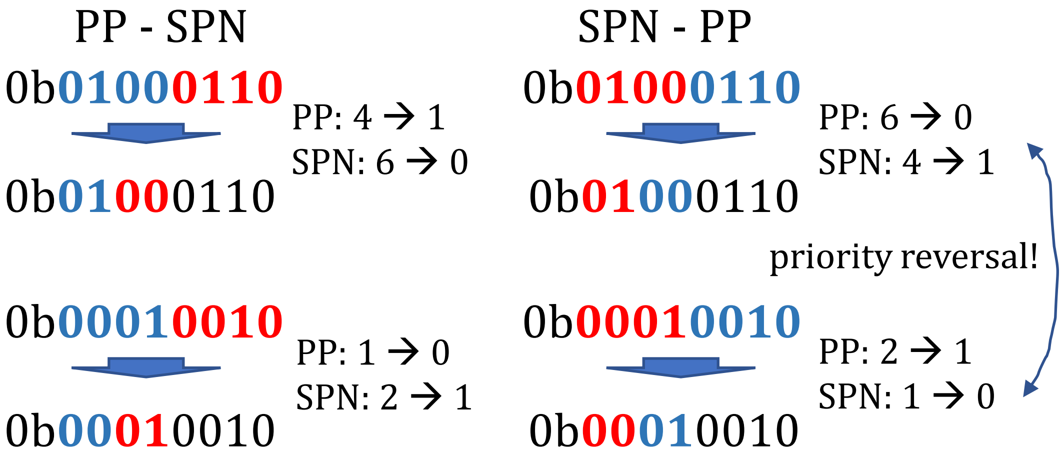
Peripheral Interrupt Priority

- Interrupt priority is configured by Interrupt Priority Register (IPR) 0~123
  - 124 IPRs * 4 priority configuration per IPR = 496 (= Total # of peripheral interrupts supported on ARMv7-M)
- Each priority consists of two fields, including preempt priority number and sub-priority number.
  - Available bits in a priority byte are device-implementation defined.
    - ranging from 3 to 8 bits
  - The preempt priority number defines the priority for preemption.
  - The sub-priority number determines the order when multiple interrupts are pending with the same preempt priority number.



Peripheral Interrupt Priority

- The lengths of the Preemption Priority-field and the Sub-priority Number-field are configurable within valid priority bits.
  - Preemption Priority-field should be located at MSB
- Why?
  - Easier porting!
  - Programs implemented on many-bit priority-level device can run on small-bit priority level device without inversion of priority.
  - example: 4/4 bits priority fields → 2/2 bits priority fields



PP - SPN

0b01000110

0b01000110

PP: 4 → 1

SPN: 6 → 0

0b00010010

0b00010010

PP: 1 → 0

SPN: 2 → 1

SPN - PP

0b01000110

0b01000110

PP: 6 → 0

SPN: 4 → 1

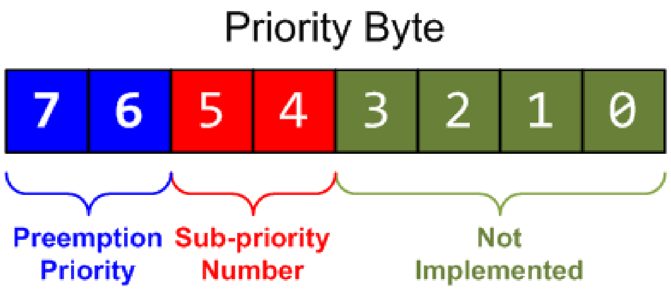
priority reversal!

0b00010010

0b00010010

PP: 2 → 1

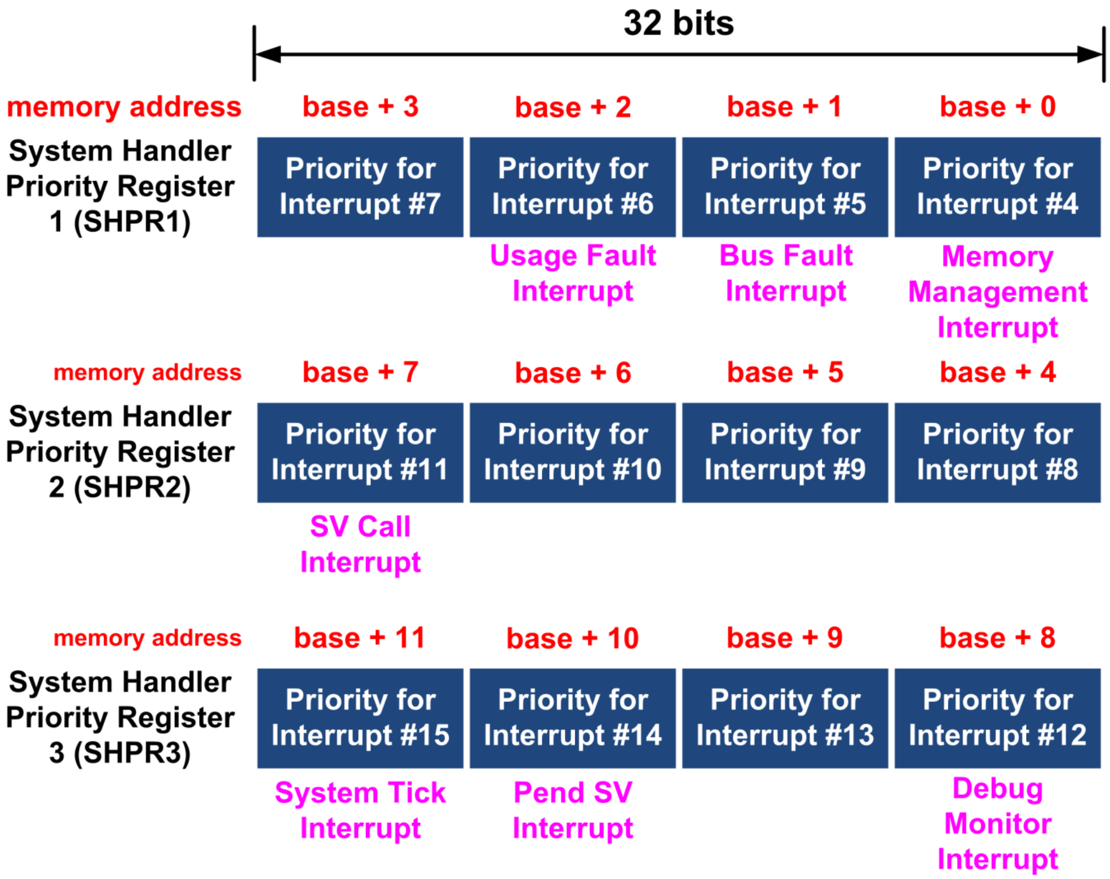
SPN: 1 → 0



default setting

System Interrupt Priority

- 3 SHPRs (System Handler Priority Register) determine priorities of system exceptions and interrupts,



32 bits

memory address	base + 3	base + 2	base + 1	base + 0
System Handler Priority Register 1 (SHPR1)	Priority for Interrupt #7	Priority for Interrupt #6	Priority for Interrupt #5	Priority for Interrupt #4
		Usage Fault Interrupt	Bus Fault Interrupt	Memory Management Interrupt
memory address	base + 7	base + 6	base + 5	base + 4
System Handler Priority Register 2 (SHPR2)	Priority for Interrupt #11	Priority for Interrupt #10	Priority for Interrupt #9	Priority for Interrupt #8
	SV Call Interrupt			
memory address	base + 11	base + 10	base + 9	base + 8
System Handler Priority Register 3 (SHPR3)	Priority for Interrupt #15	Priority for Interrupt #14	Priority for Interrupt #13	Priority for Interrupt #12
	System Tick Interrupt	Pend SV Interrupt		Debug Monitor Interrupt

Exception-masking registers

- PRIMASK: Set current execution priority to 0
  - All interrupts are disabled except for NMI, hard fault, and reset

```
MOV R0, #1
MSR PRIMASK, R0
```

- FAULTMASK: Set current execution priority to -1
  - All interrupts are disabled except for NMI and reset
- BASEPRI: Disable all interrupts of the same or lower priority level
  - Example
    - disabling interrupts whose priority levels are equal to or lower than 0x60

```
MOV R0, #0x60
MSR BASEPRI, R0
```

Summary

- Interrupt handling process
  - stacking/unstacking
  - exception return
- Priority management