Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

ARM Cortex-M
Interrupt

Z. Gu

Fall 2025

Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine https://web.eece.maine.edu/~zhu/book/

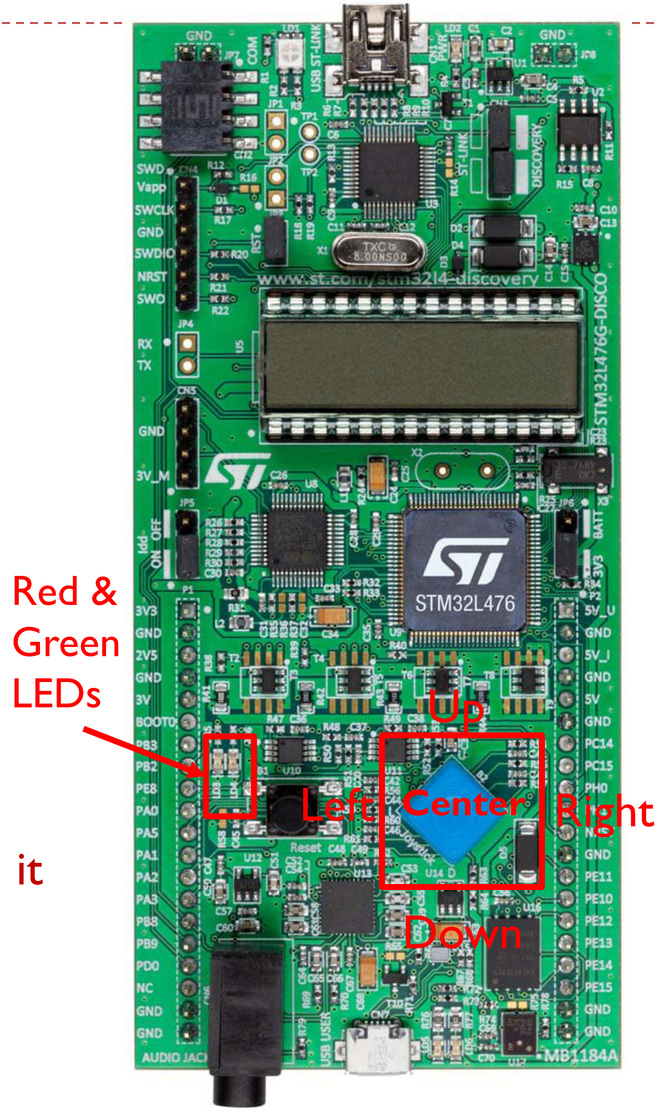
Polling *vs* Interrupt

Copyright © Ron Leishman * http://ToonClips.com/9845

Polling:
You pick up the phone every few seconds to check whether you are getting a call.

Interrupt:
Do whatever you should do and pick up the phone when it rings.

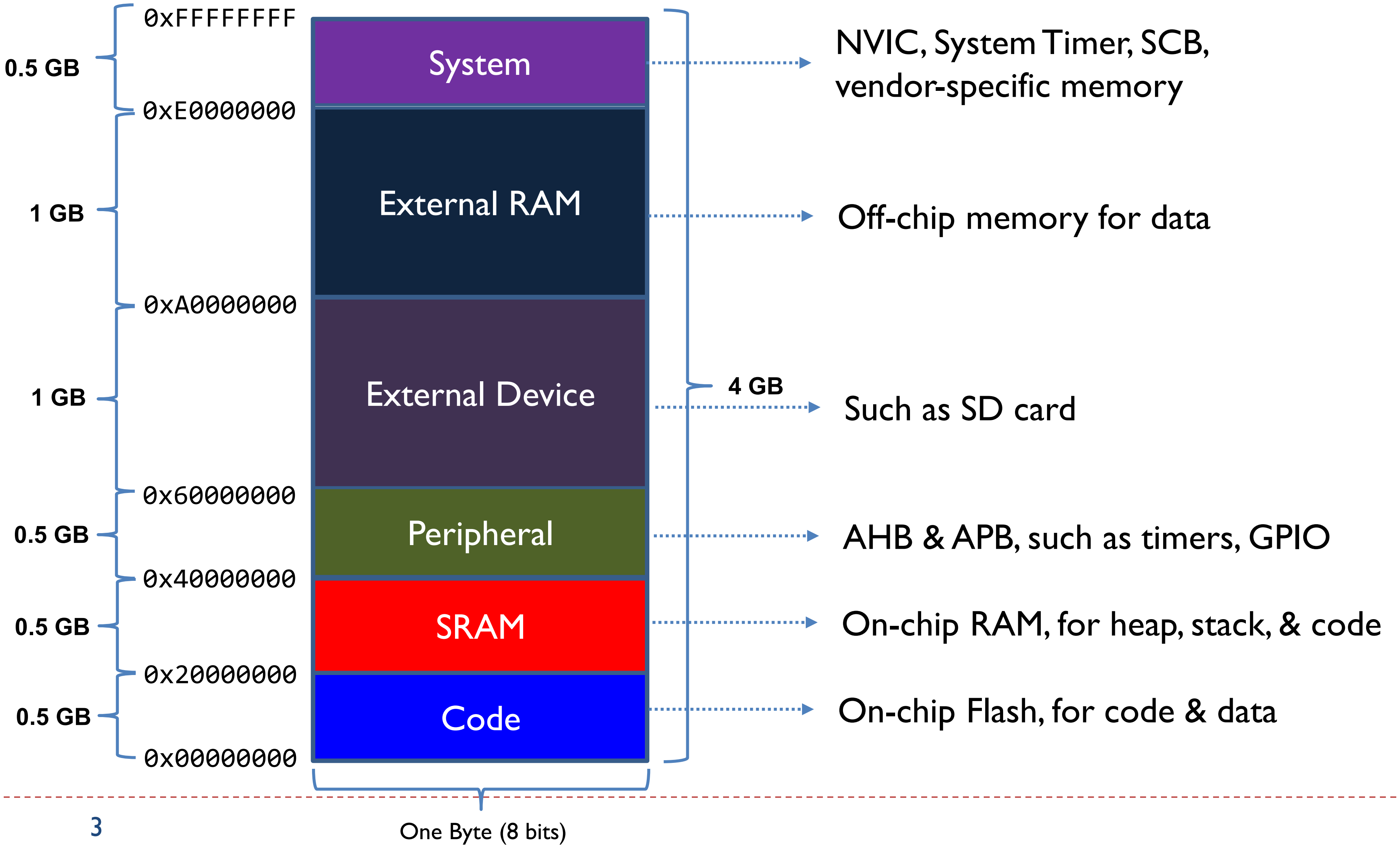
STM32L4 Discovery Kit



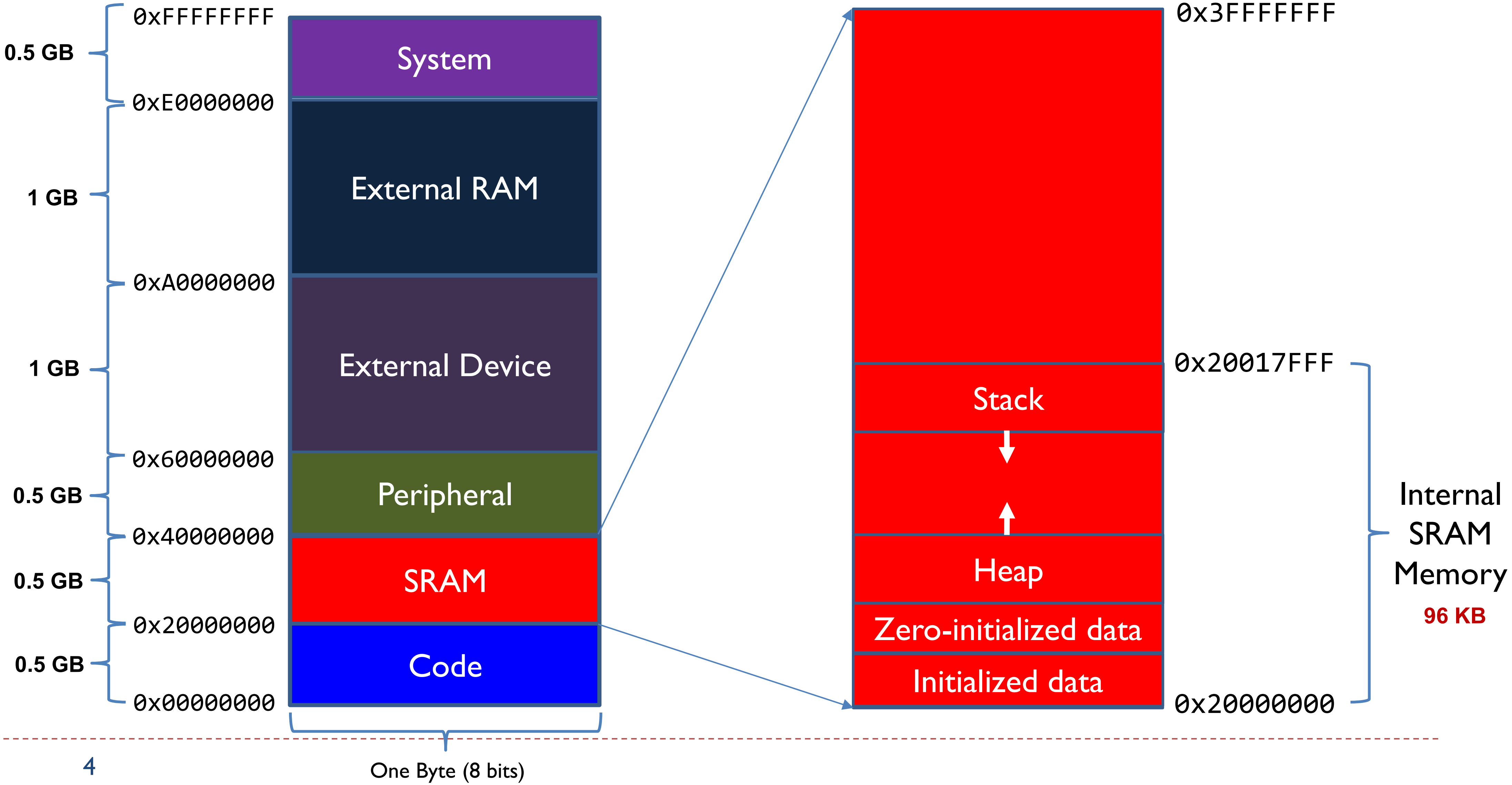
```
// Polling method
while (1) {
     read_button_input;
     if (pushed)
          exit;
}

turn_on_LED;
```

```
// Interrupt method
interrupt_handler(){
 turn_on_LED;
 exit;
}
```

Memory Map of Cortex-M4

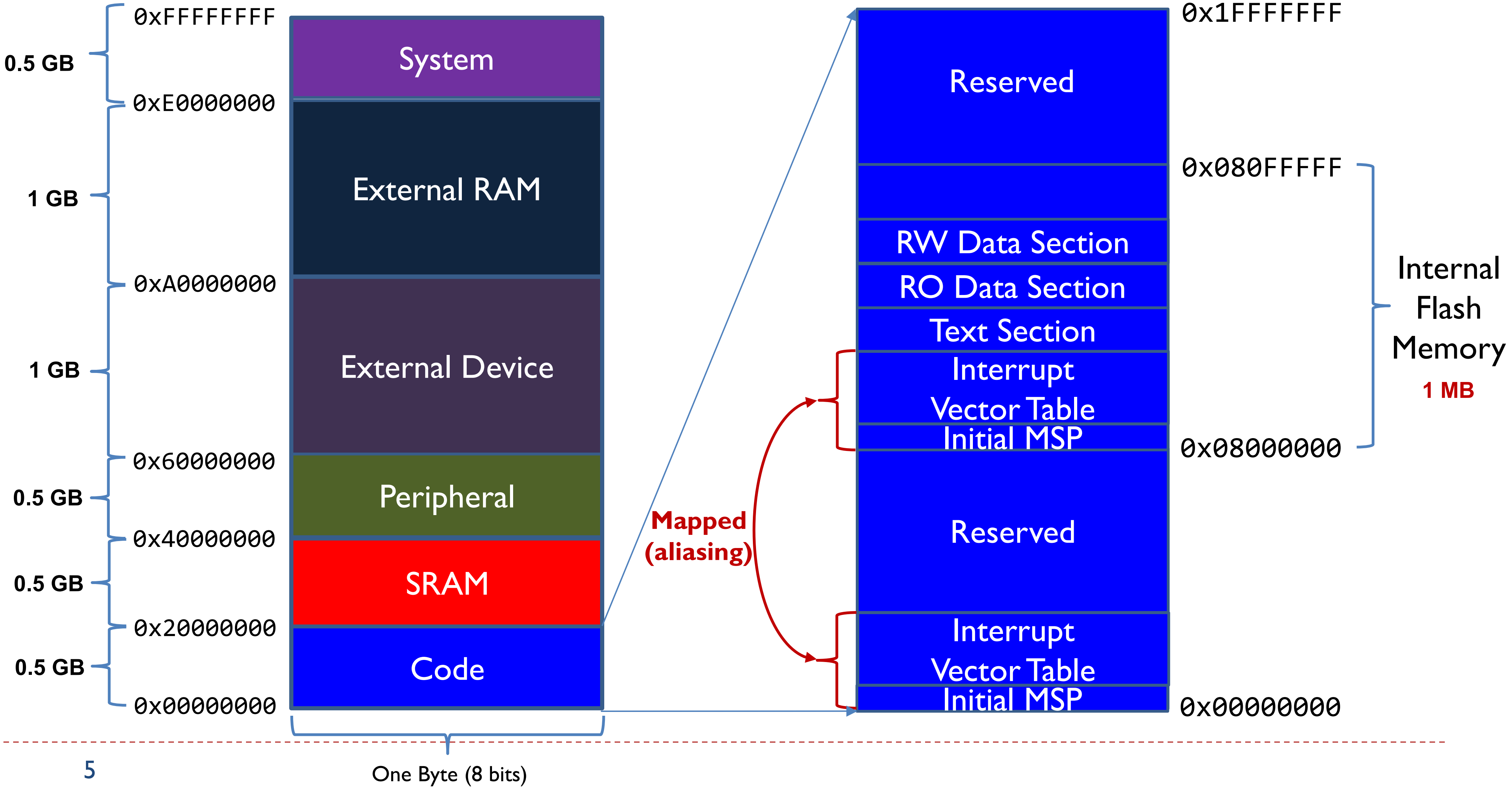


Data Memory

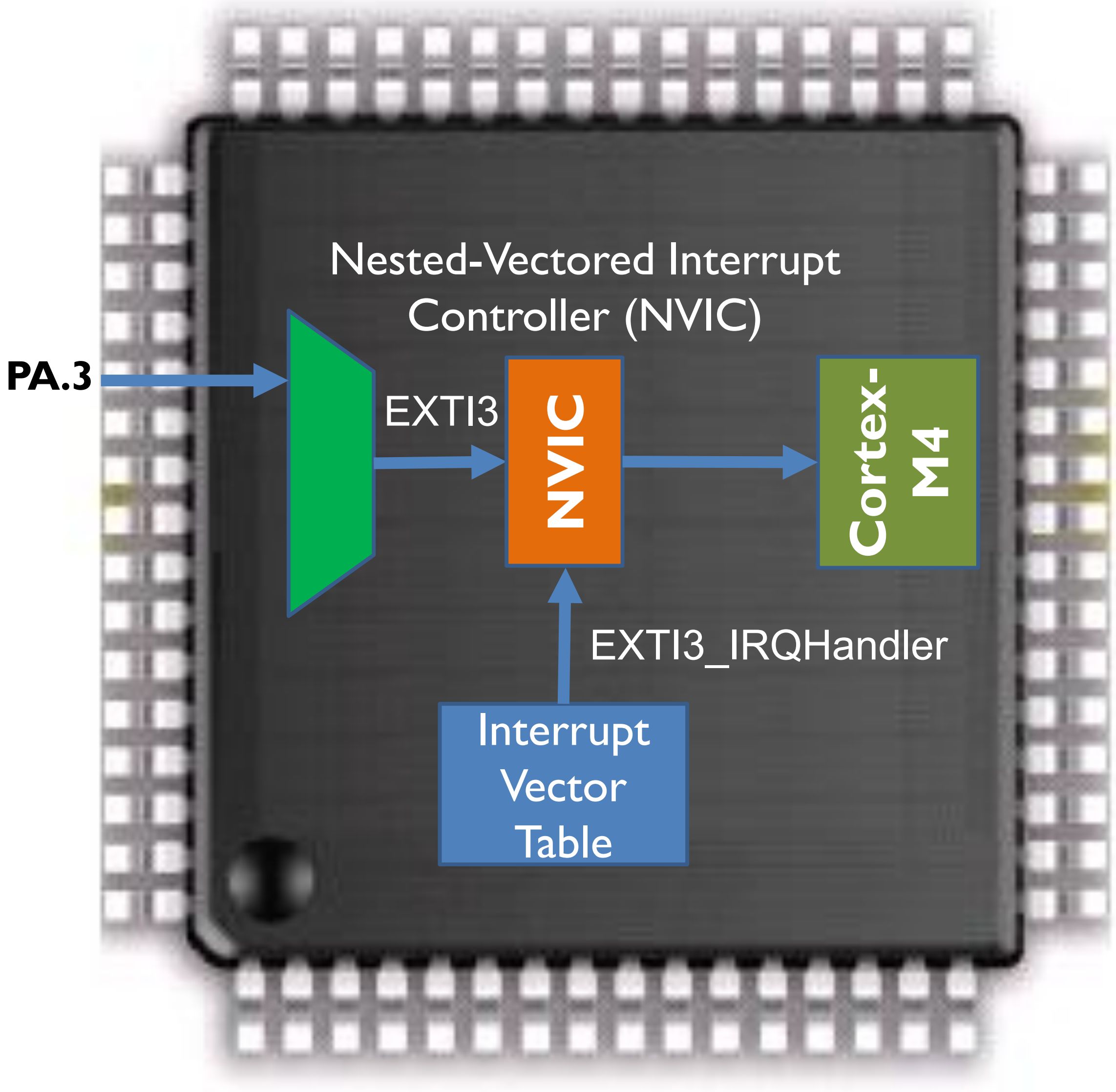


0.5 GB
1 GB
1 GB
0.5 GB
0.5 GB
0.5 GB
0xFFFFFFFF
0xE0000000
0xA0000000
0x60000000
0x40000000
0x20000000
0x00000000
System
External RAM
External Device
Peripheral
SRAM
Code
One Byte (8 bits)
0x3FFFFFFF
0x20017FFF
0x20000000
Stack
Heap
Zero-initialized data
Initialized data
Internal SRAM Memory
96 KB

Instruction Memory



Interrupt Vector Table



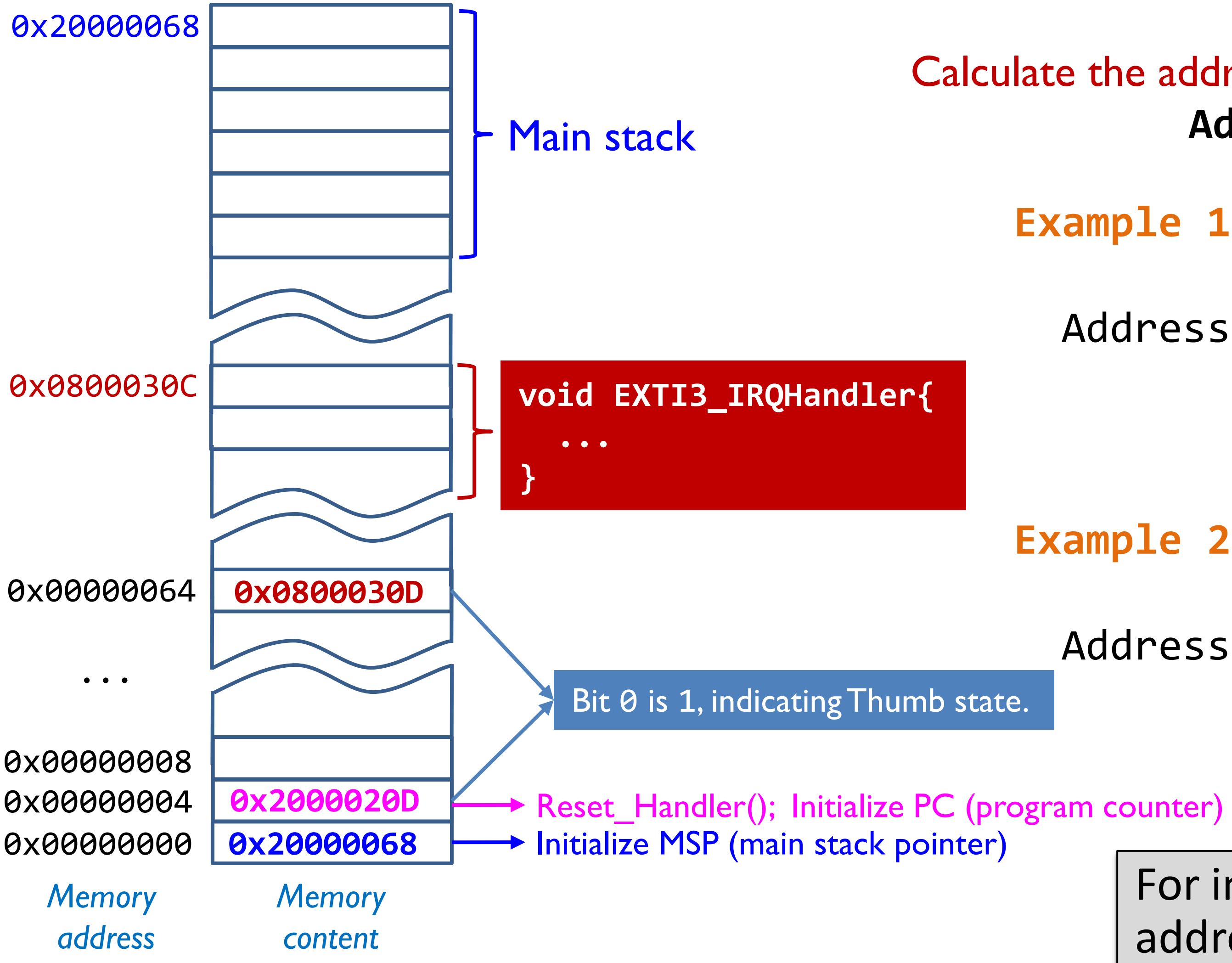
Interrupt Vector Table

Address of ISR 1

Interrupt Number (8 bits)	Memory Address of ISR (32 bits)
1	Interrupt Service Routine for interrupt 1
2	Interrupt Service Routine for interrupt 2
3	Interrupt Service Routine for interrupt 3
4	Interrupt Service Routine for interrupt 4
5	Interrupt Service Routine for interrupt 5
…	…

When interrupt x is triggered, jump to the ISR for interrupt x. (1 ≤ x ≤ 255)

Interrupt Vector Table



```
void EXTI3_IRQHandler{
  ...
}
```

Bit 0 is 1, indicating Thumb state.

Reset_Handler(); Initialize PC (program counter)

Initialize MSP (main stack pointer)

Calculate the address which holds the address of the ISR for interrupt n:

Address of pointer = 64 + 4 × n

Example 1: EXTI3_IRQn = 9

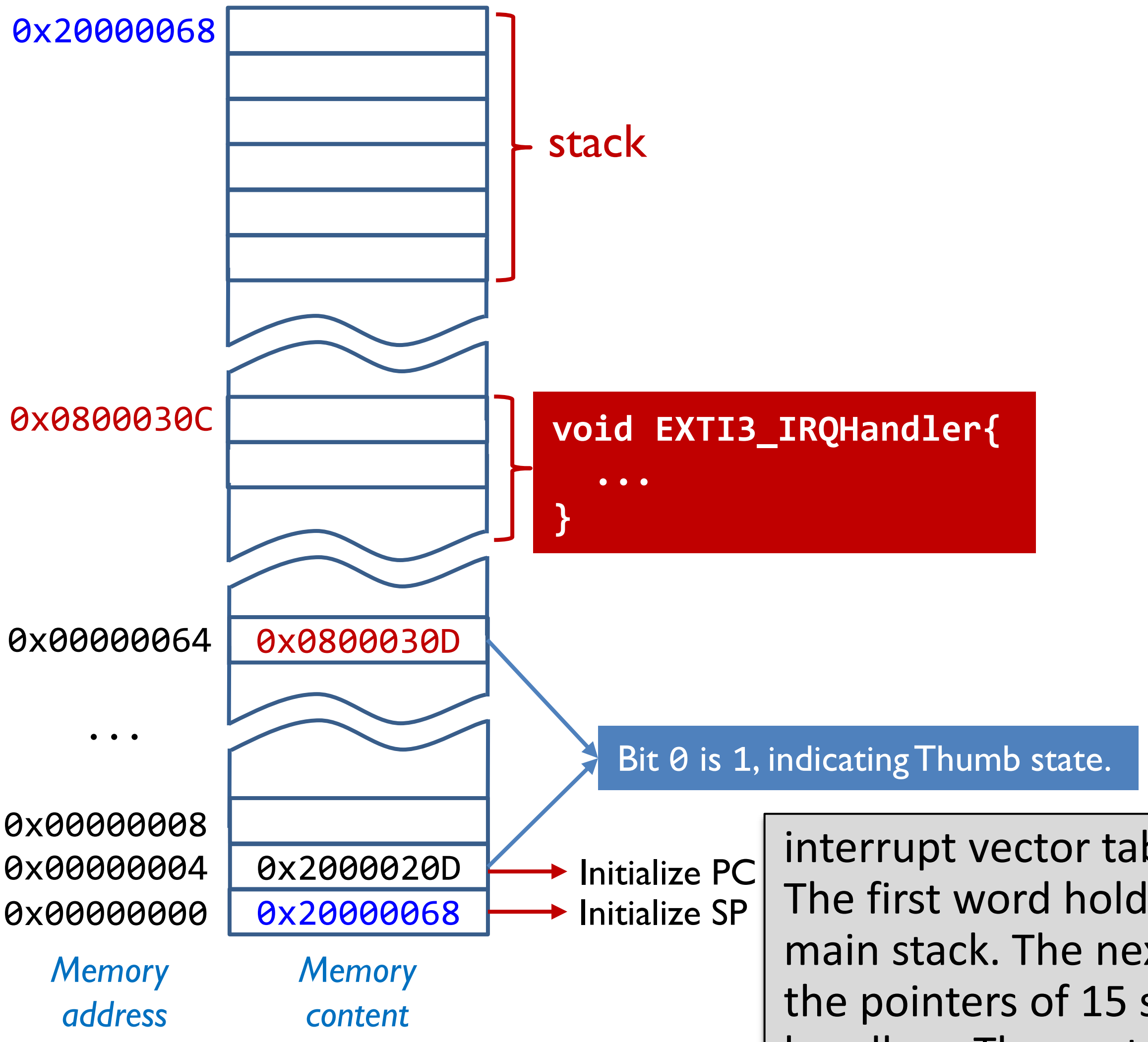
```
Address of pointer to EXTI3 ISR = 64 + 4 × 9
                                = 100
                                = 0x64
```

Example 2: SysTick_IRQn = -1

```
Address of pointer to SysTick ISR = 64 + 4 × (-1)
                                  = 60
                                  = 0x3C
```

For interrupt number n, its ISR is stored at address 64 + 4 x n, in the interrupt vector table.

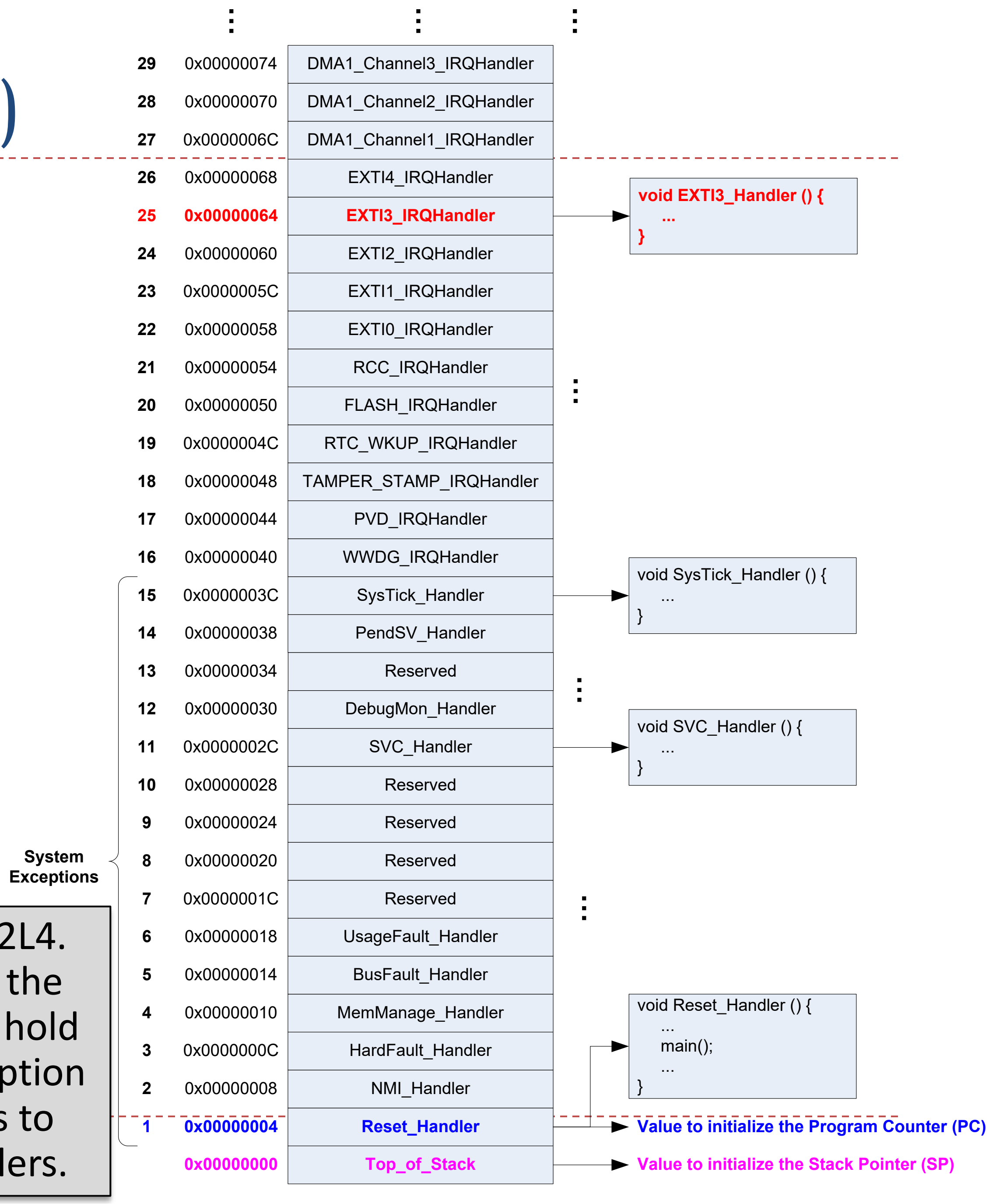
Interrupt Service Routine (ISR)



```
void EXTI3_IRQHandler{
  ...
}
```

Bit 0 is 1, indicating Thumb state.

interrupt vector table of STM32L4. The first word holds the top of the main stack. The next 15 words hold the pointers of 15 system exception handlers. The next are pointers to vender-specific interrupt handlers.



⋮	⋮	⋮
29	0x00000074	DMA1_Channel3_IRQHandler
28	0x00000070	DMA1_Channel2_IRQHandler
27	0x0000006C	DMA1_Channel1_IRQHandler
26	0x00000068	EXTI4_IRQHandler
25	0x00000064	EXTI3_IRQHandler
24	0x00000060	EXTI2_IRQHandler
23	0x0000005C	EXTI1_IRQHandler
22	0x00000058	EXTI0_IRQHandler
21	0x00000054	RCC_IRQHandler
20	0x00000050	FLASH_IRQHandler
19	0x0000004C	RTC_WKUP_IRQHandler
18	0x00000048	TAMPER_STAMP_IRQHandler
17	0x00000044	PVD_IRQHandler
16	0x00000040	WWDG_IRQHandler
15	0x0000003C	SysTick_Handler
14	0x00000038	PendSV_Handler
13	0x00000034	Reserved
12	0x00000030	DebugMon_Handler
11	0x0000002C	SVC_Handler
10	0x00000028	Reserved
9	0x00000024	Reserved
8	0x00000020	Reserved
7	0x0000001C	Reserved
6	0x00000018	UsageFault_Handler
5	0x00000014	BusFault_Handler
4	0x00000010	MemManage_Handler
3	0x0000000C	HardFault_Handler
2	0x00000008	NMI_Handler
1	0x00000004	Reset_Handler
	0x00000000	Top_of_Stack

```
void EXTI3_Handler () {
  ...
}
```

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

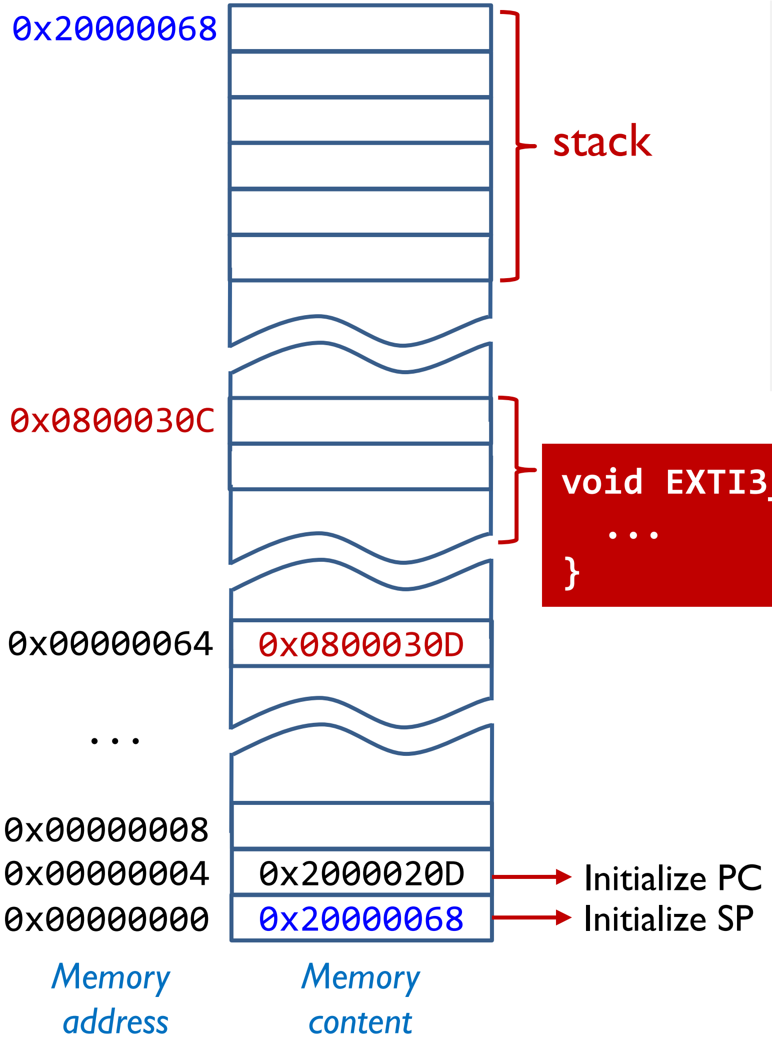
```
void SVC_Handler () {
  ...
}
```

```
void Reset_Handler () {
  ...
  main();
  ...
}
```

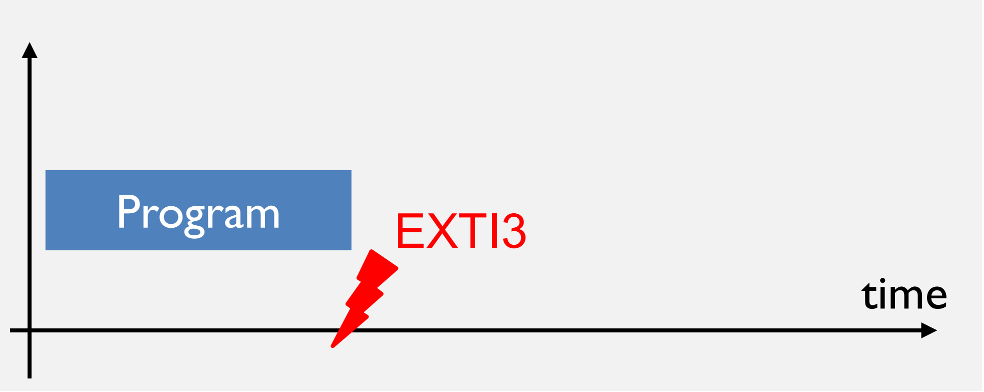
Value to initialize the Program Counter (PC)

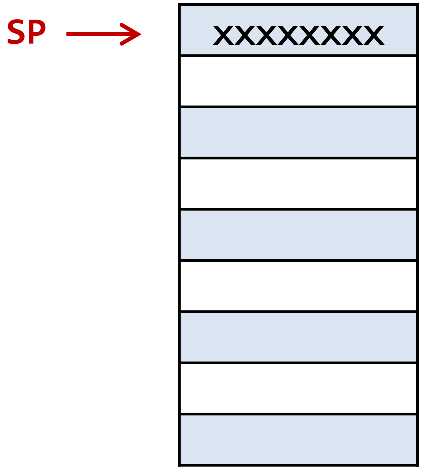
Value to initialize the Stack Pointer (SP)

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

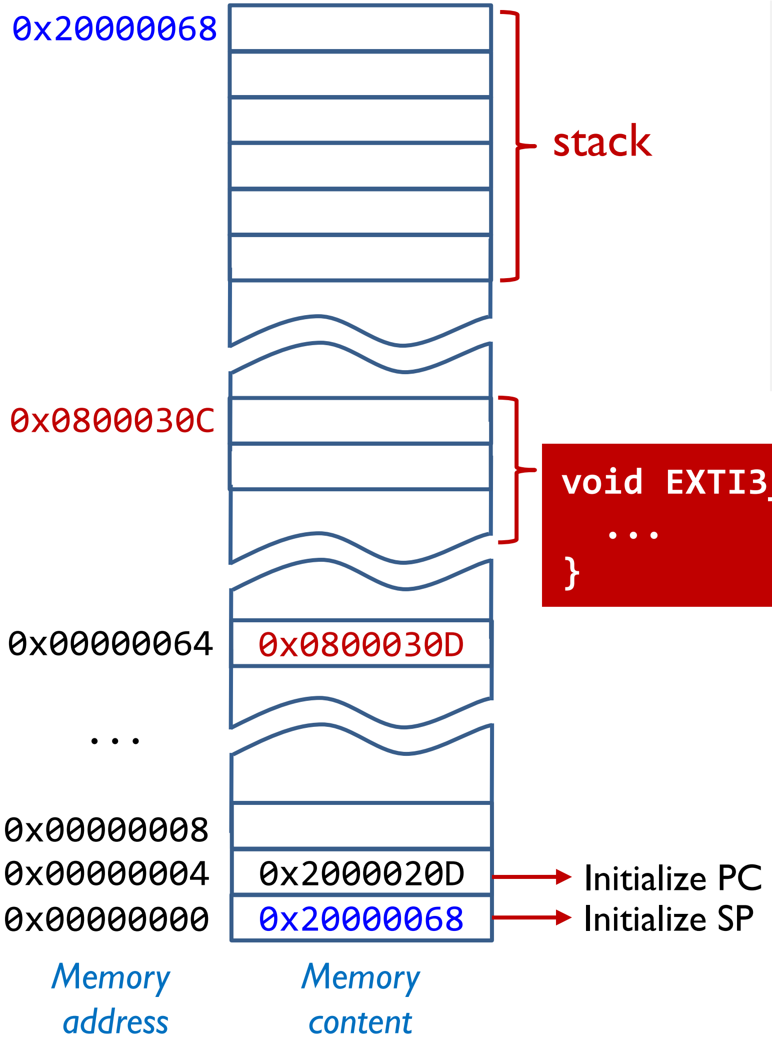




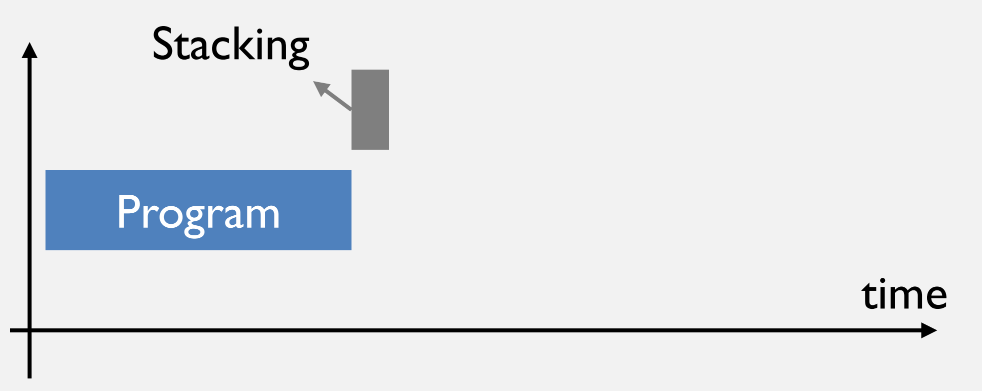
Suppose EXTI3 arrives. Assume software has already enabled EXTI3 by setting the corresponding bit in the interrupt Enable Register. The interrupt number of EXTI3 is pre-defined as 9. Interrupt priority register shows that its priority is set to 2.

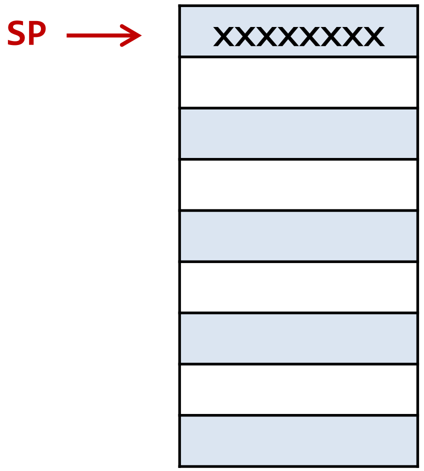
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

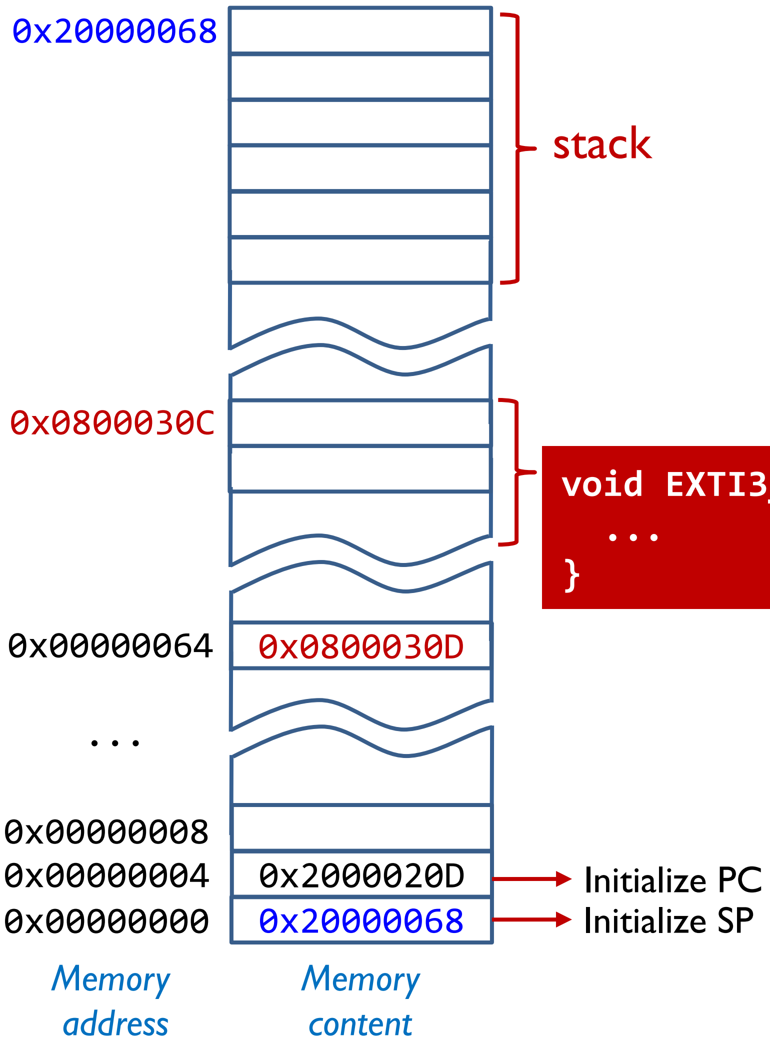




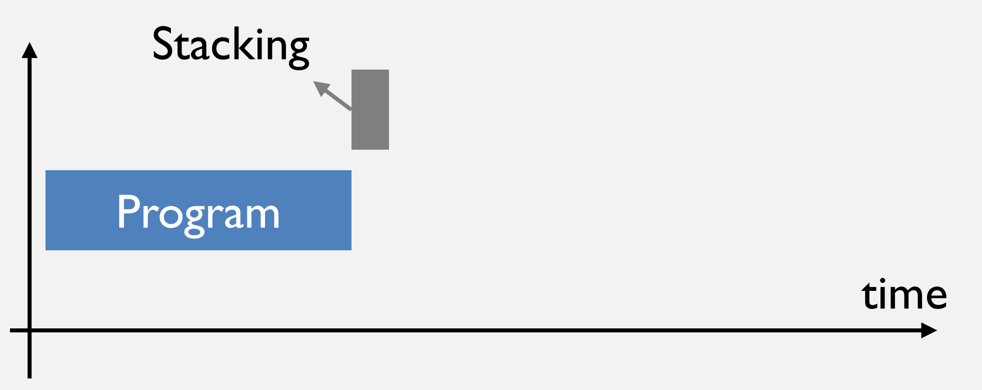
NVIC first writes 1 to the pending bit of EXTI3 in the interrupt pending register. NVIC starts to the stacking process, and pushes, 8 registers to preserve the running environment. (If FPU is used, more registers are pushed into the stack during the stacking.)

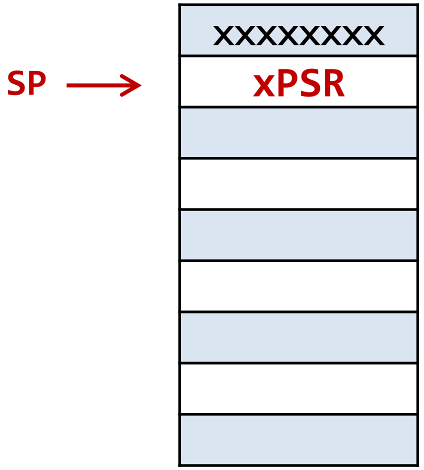
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

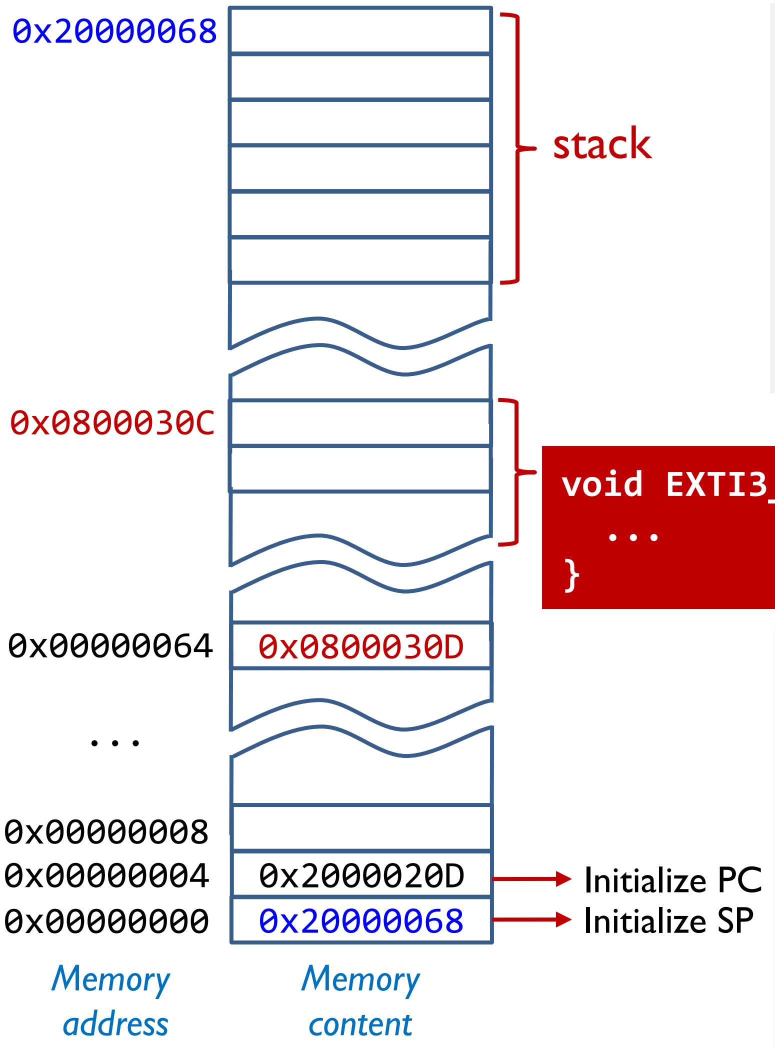




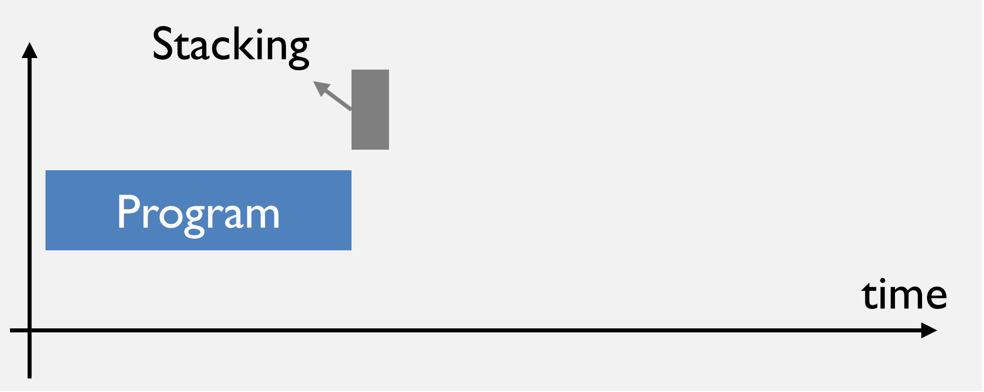
NVIC first pushes program status register (xPSR) onto the stack.

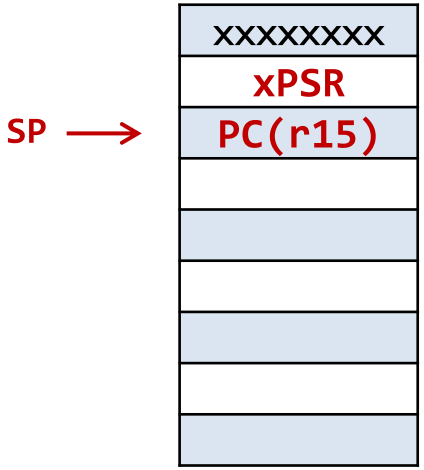
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

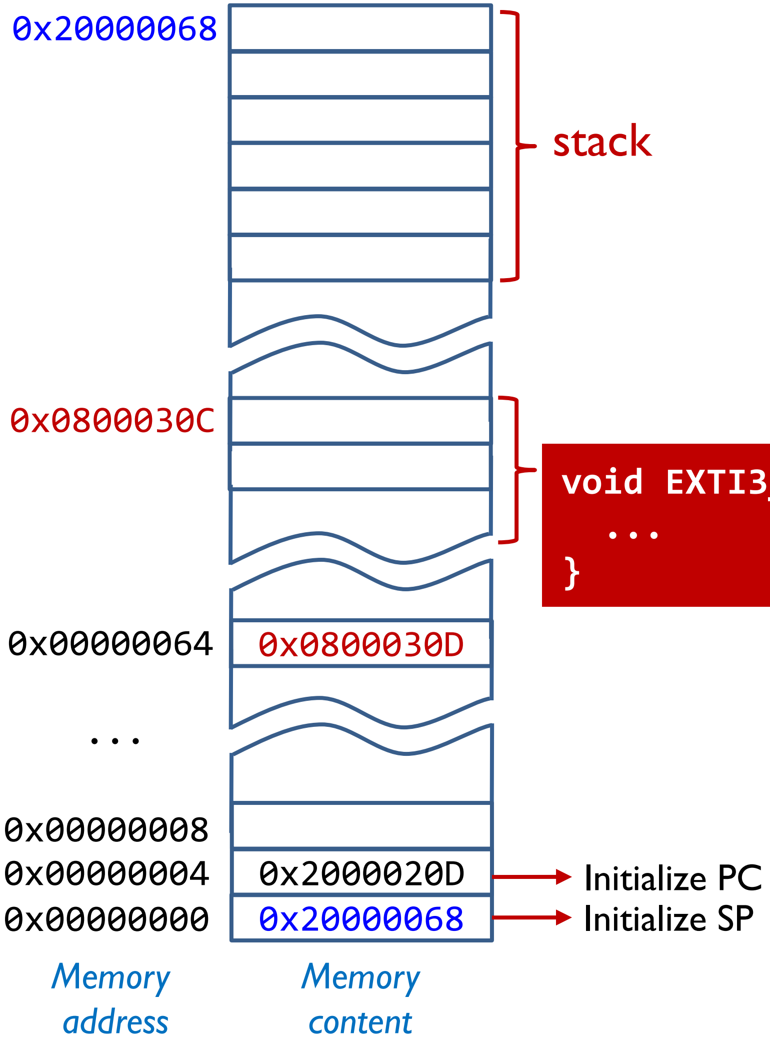




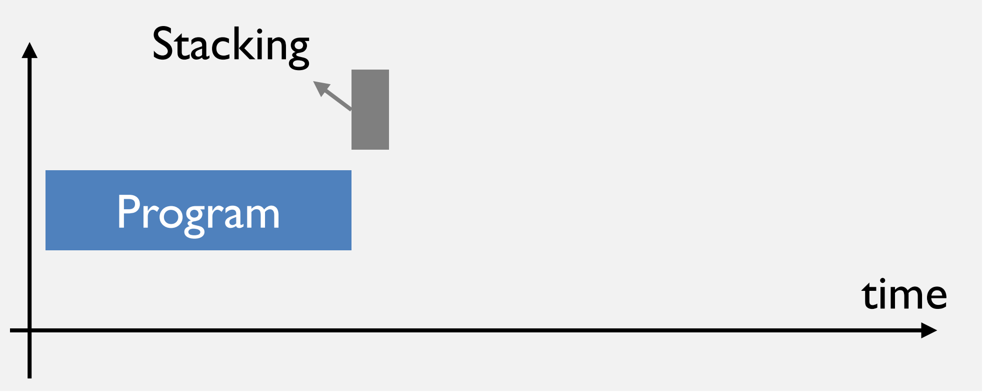
NVIC then pushes program counter (PC), which is r15.

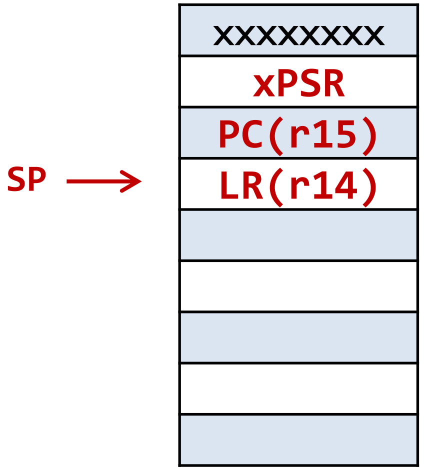
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
   ...
}
```

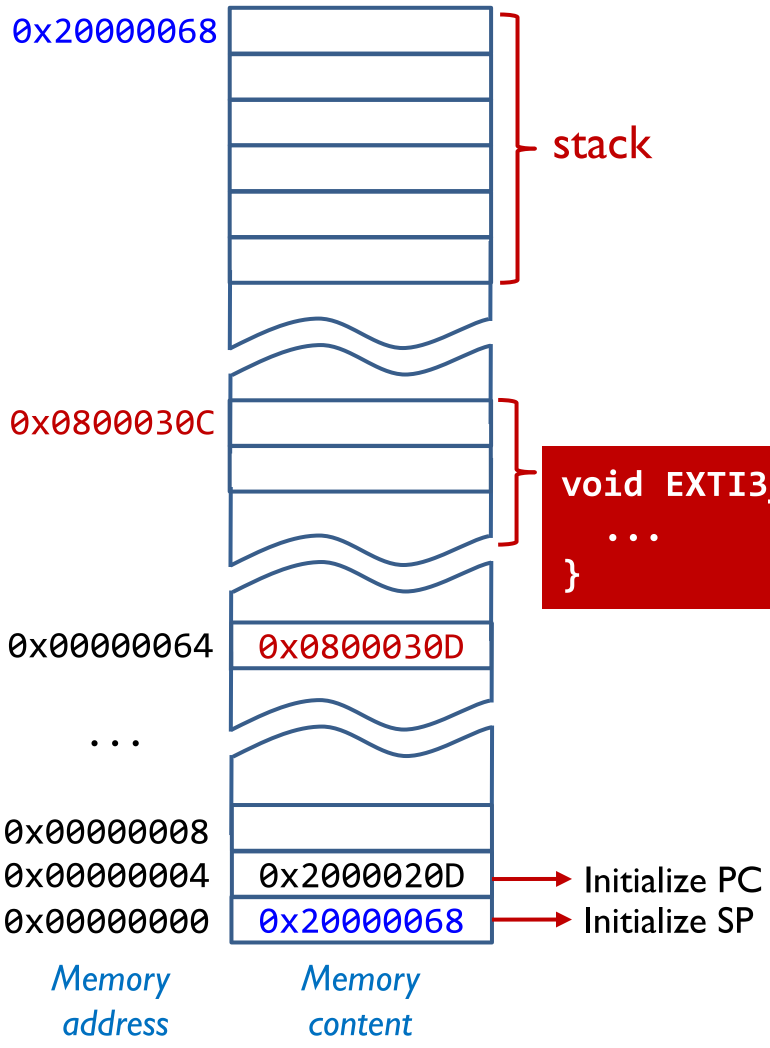




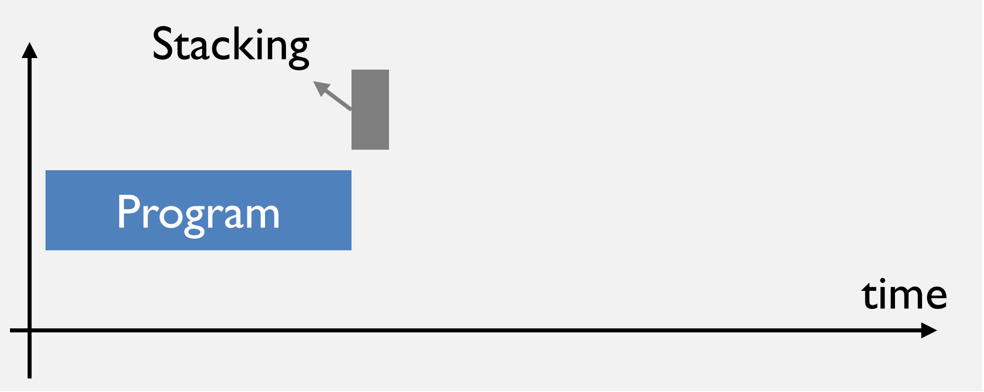
NVIC then pushes link register (LR), which is r14.

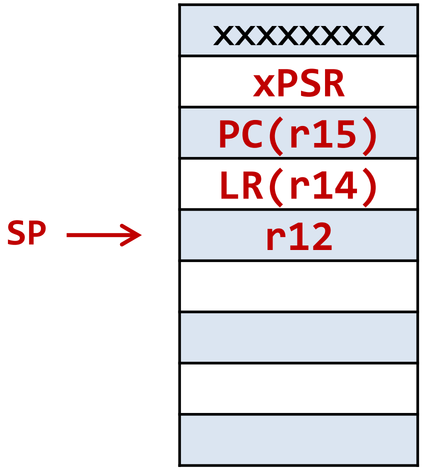
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

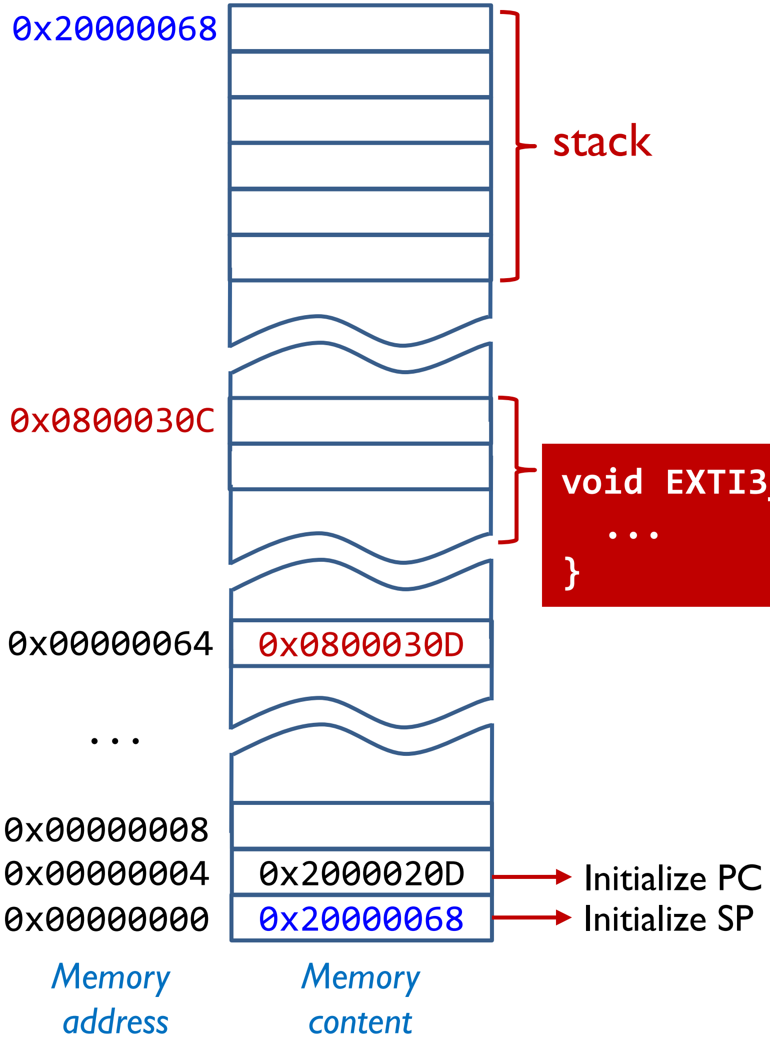




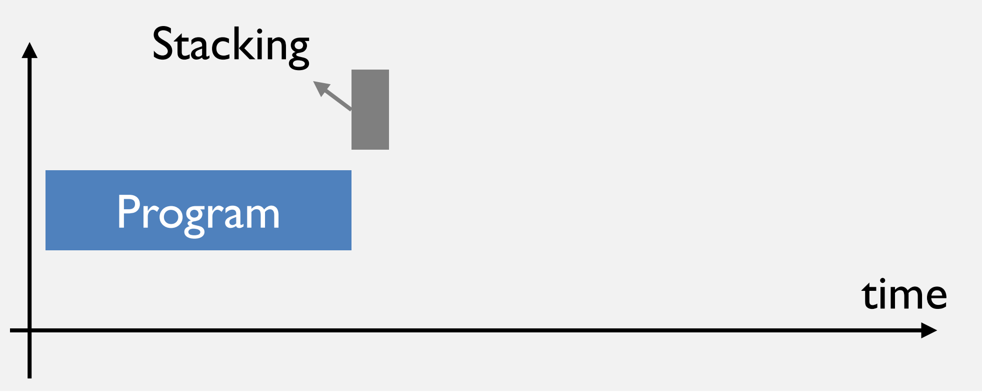
NVIC pushes r12 (intra-procedure call scratch register).

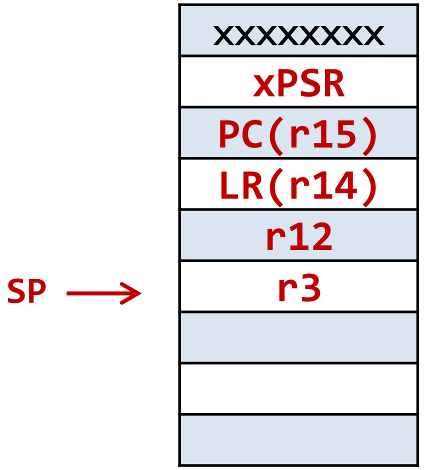
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

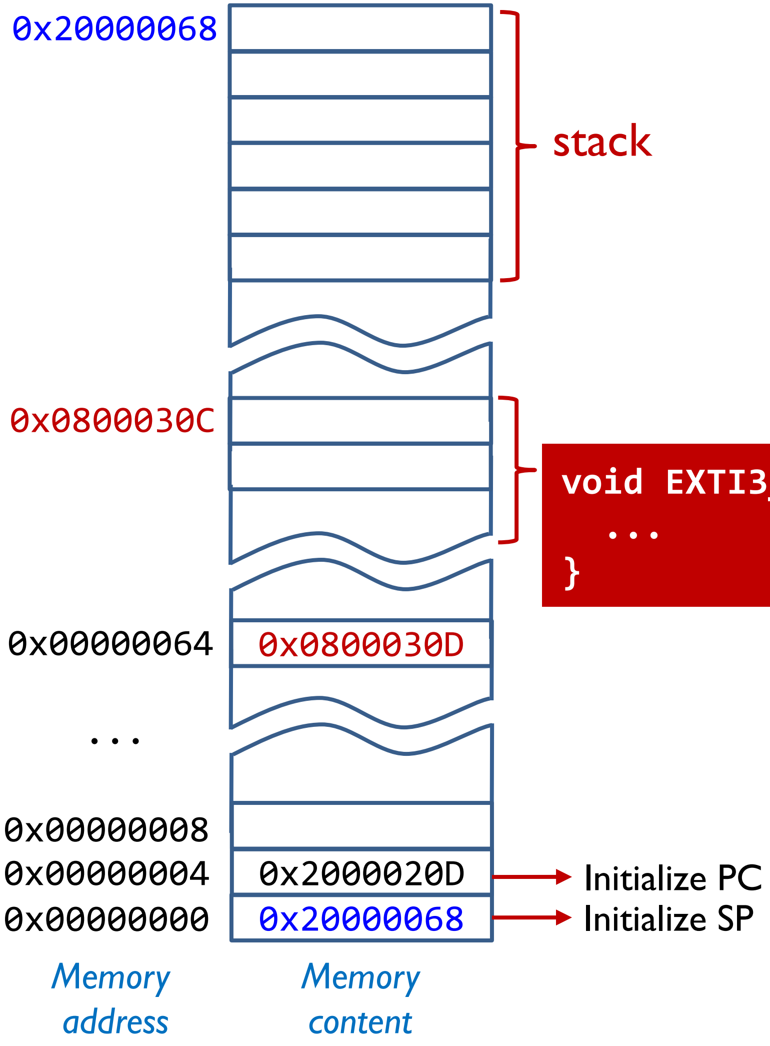




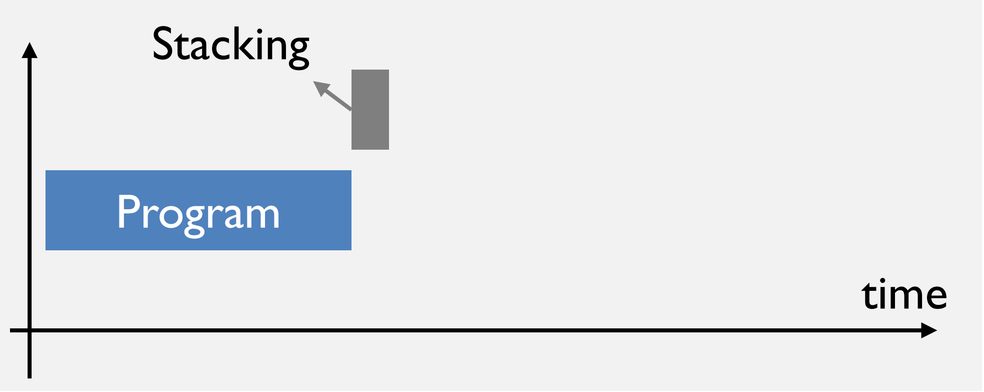
NVIC pushes r3.

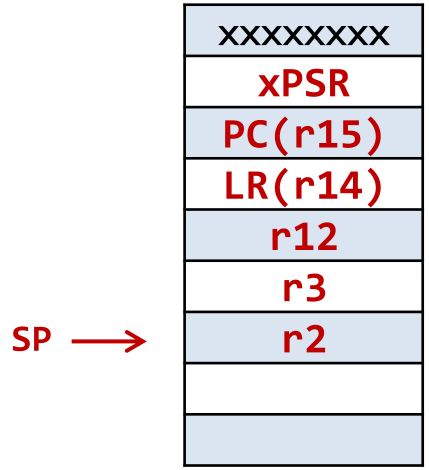
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

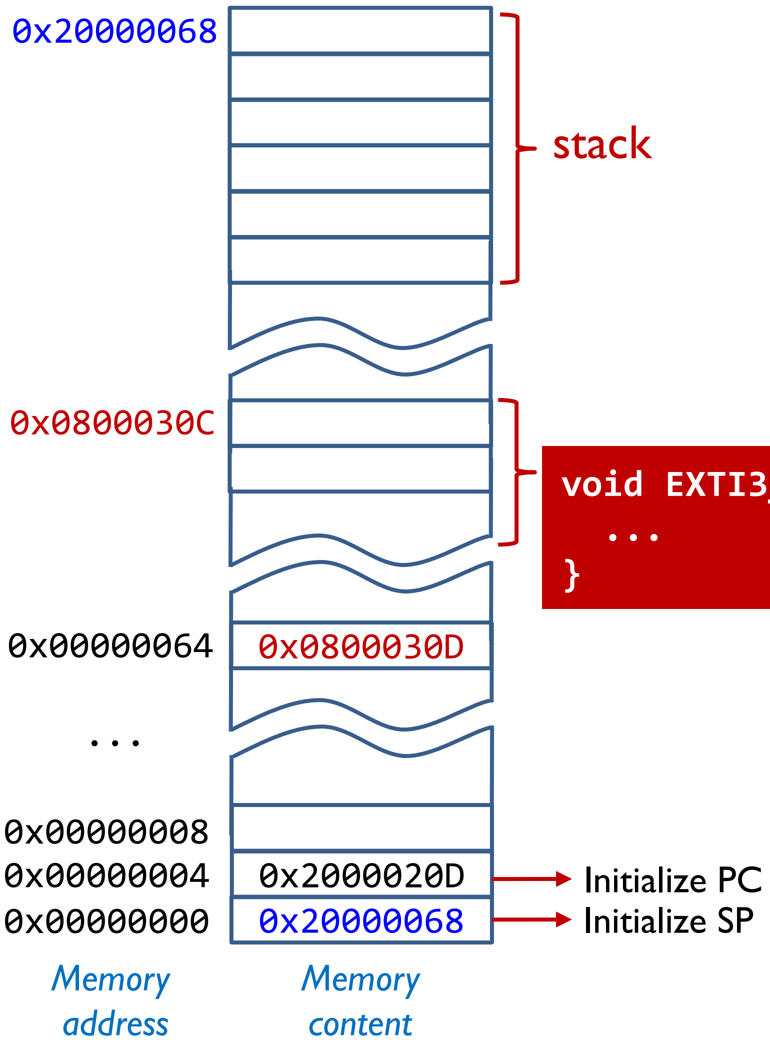




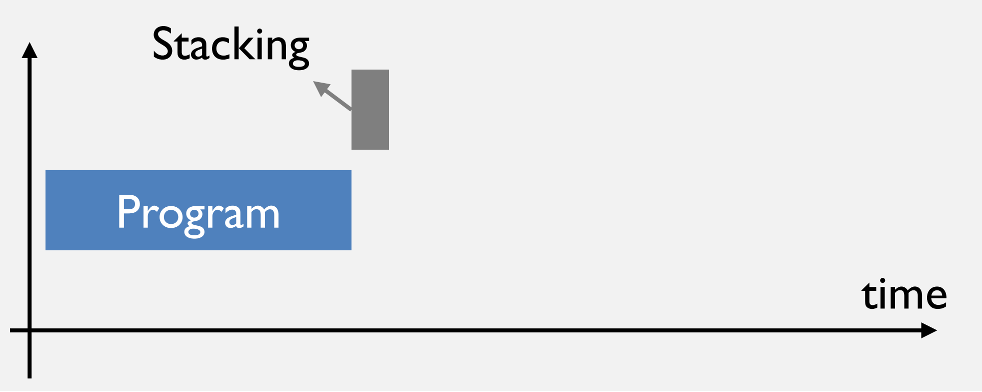
NVIC pushes r2.

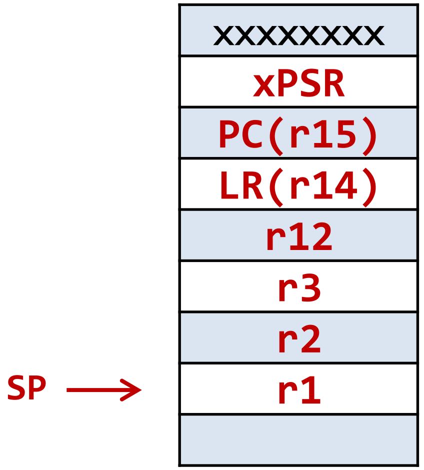
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

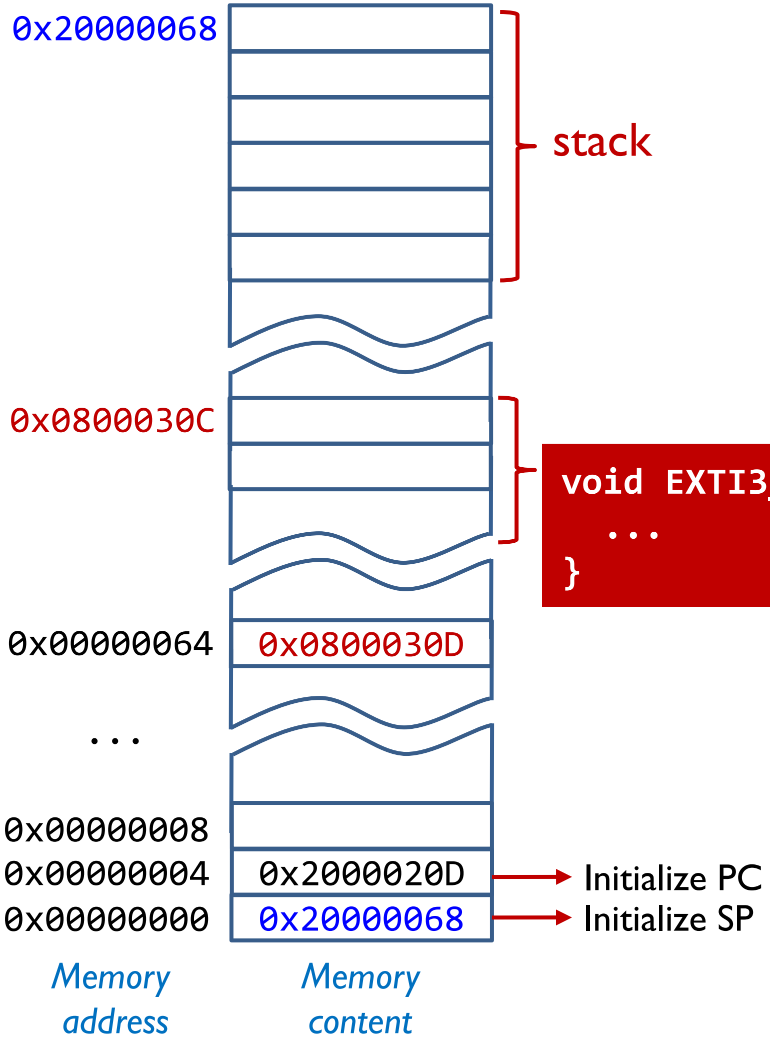




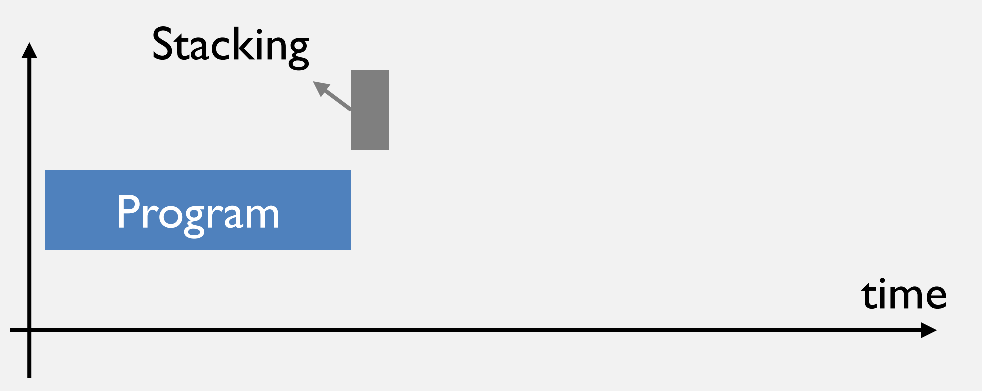
NVIC pushes r1.

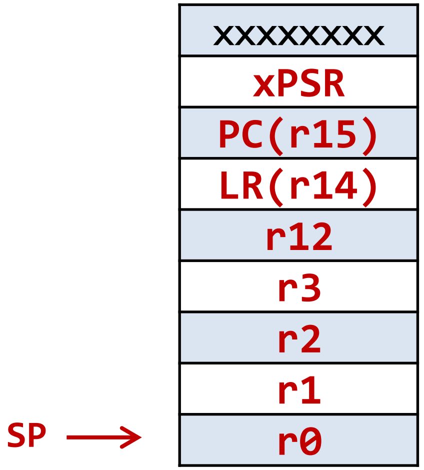
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
   ...
}
```



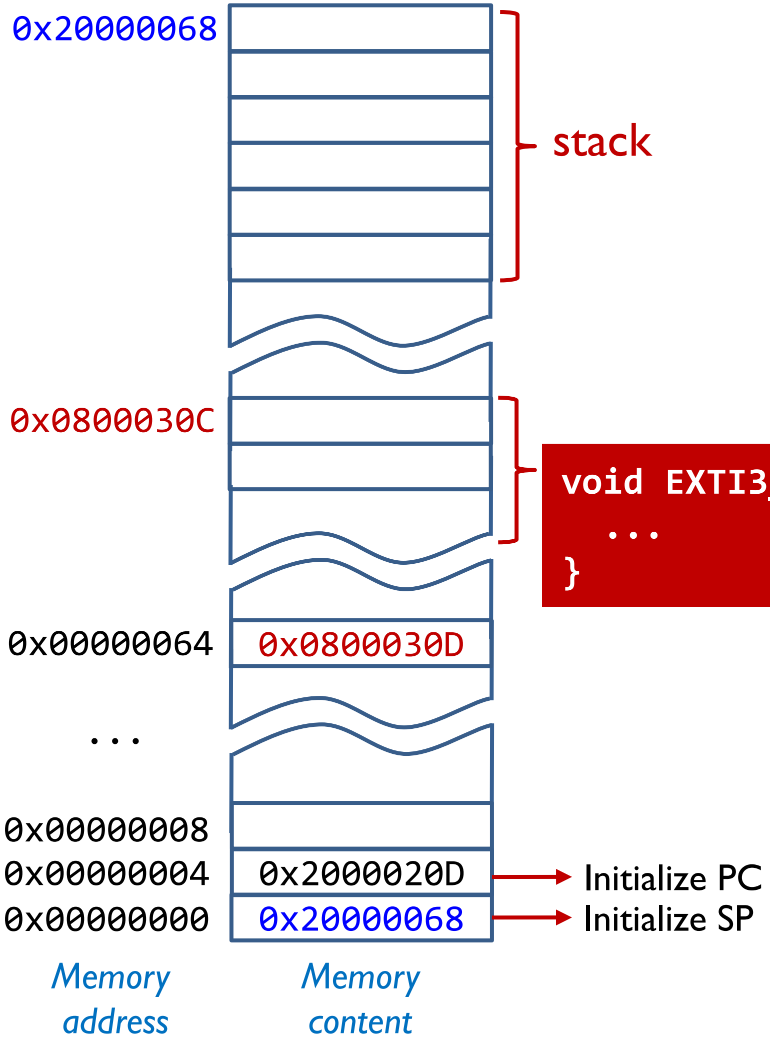


xxxxxxxx
xPSR
PC(r15)
LR(r14)
r12
r3
r2
r1
SP → r0

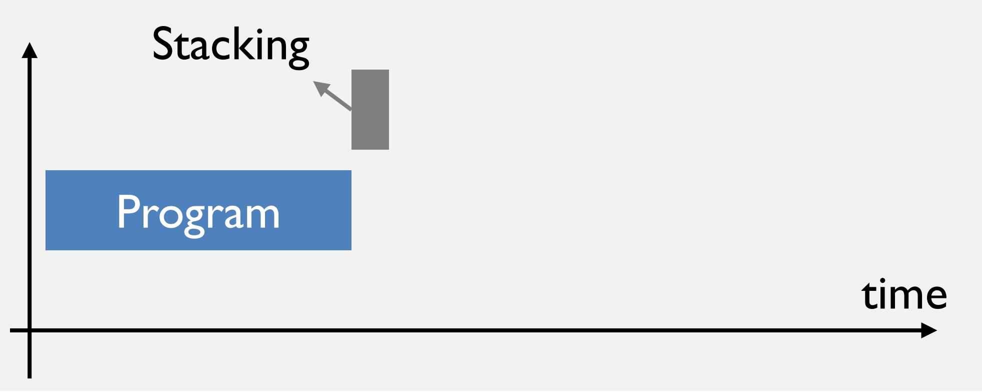
NVIC pushes r0.

				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```



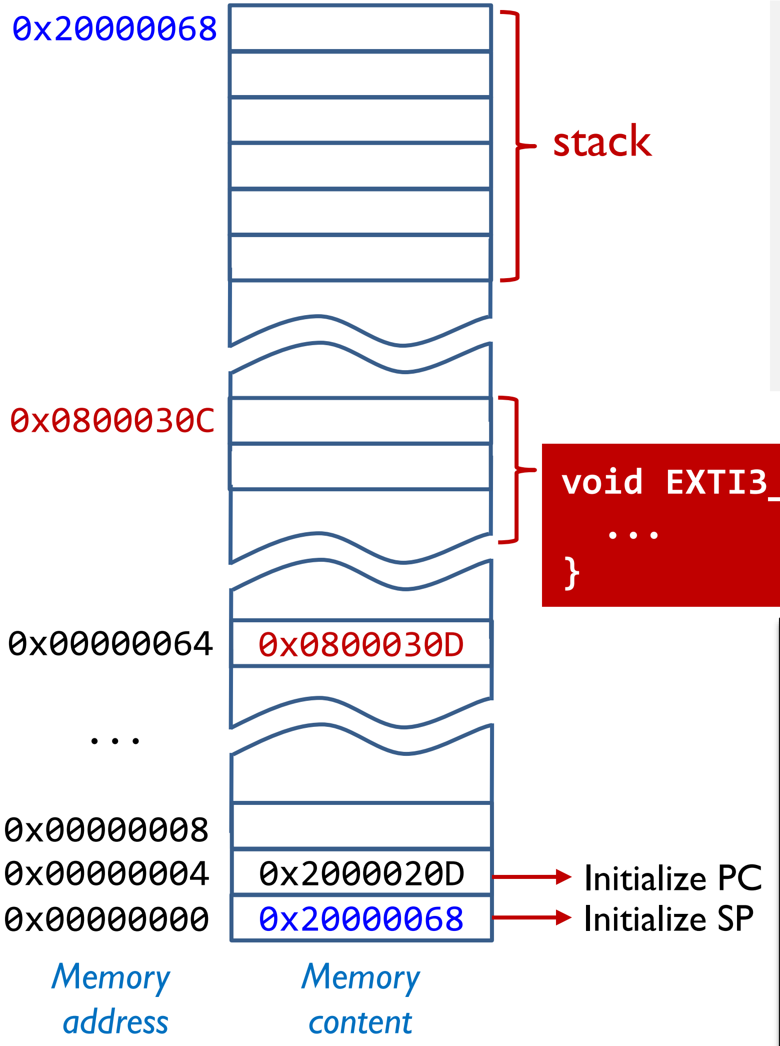
Full Descending Stack

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

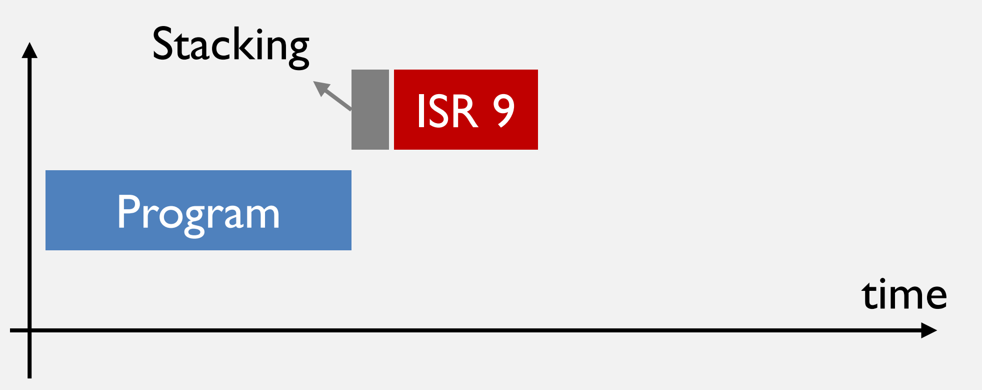
After NVIC pushes these 8 registers onto the stack, the size of the stack increases by 4*8=32 bytes, and the stack pointer is decremented by 32. (ARM Cortex uses full descending stacks.)

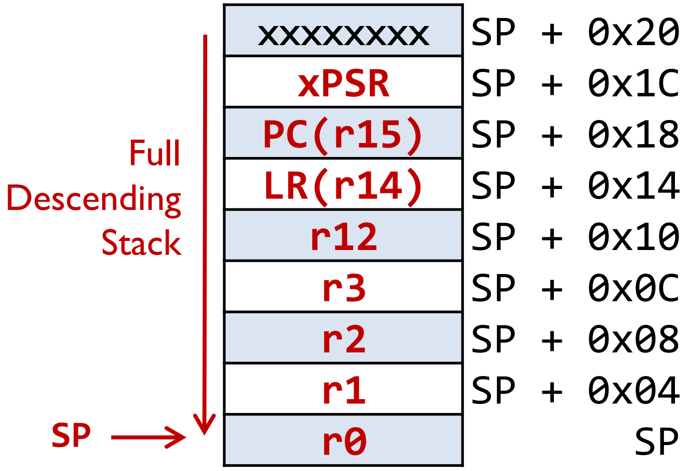
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	1	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```



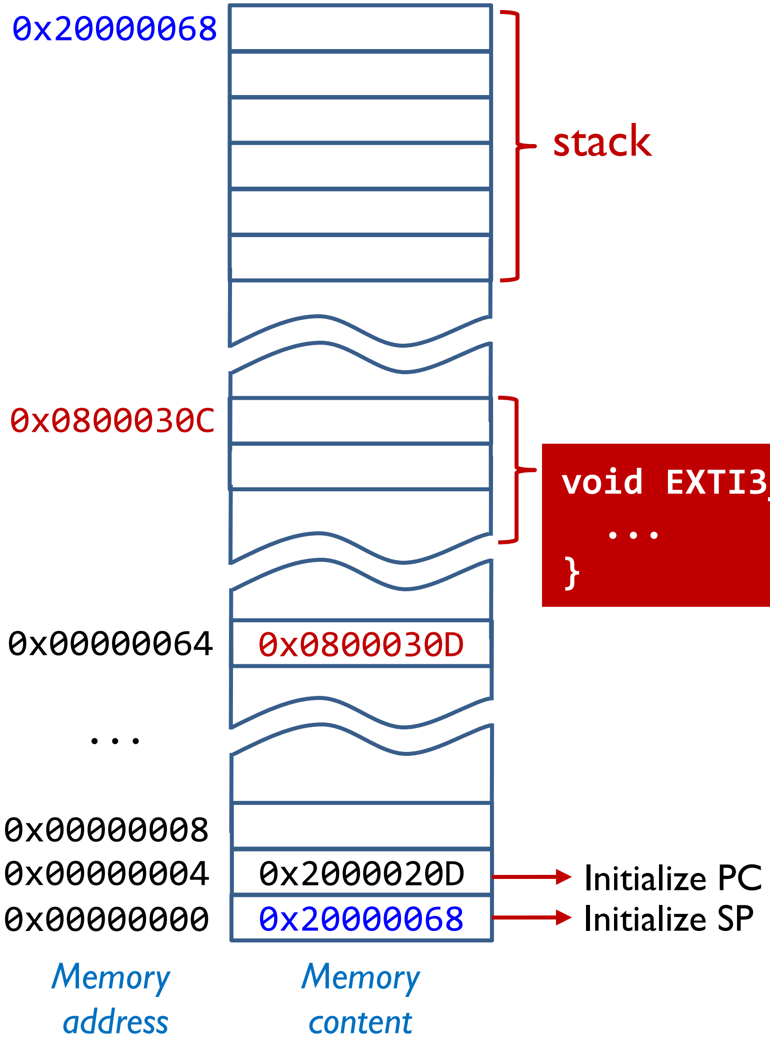


xxxxxxxx	SP + 0x20
xPSR	SP + 0x1C
PC(r15)	SP + 0x18
LR(r14)	SP + 0x14
r12	SP + 0x10
r3	SP + 0x0C
r2	SP + 0x08
r1	SP + 0x04
r0	SP

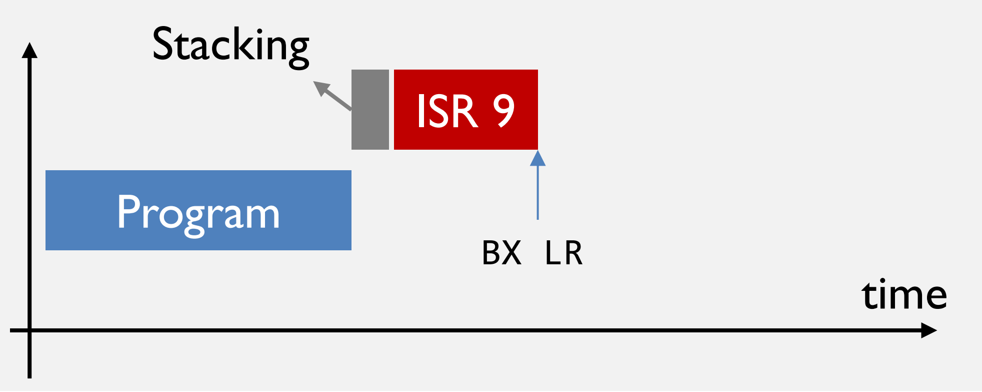
NVIC looks up interrupt vector table and finds the starting address of ISR 9 for EXTI3. It changes status of interrupt 9, from pending to active state by setting its active bit to 1 in Active Register. It then forces the processor to branch to and start to execute ISR 9.

				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```



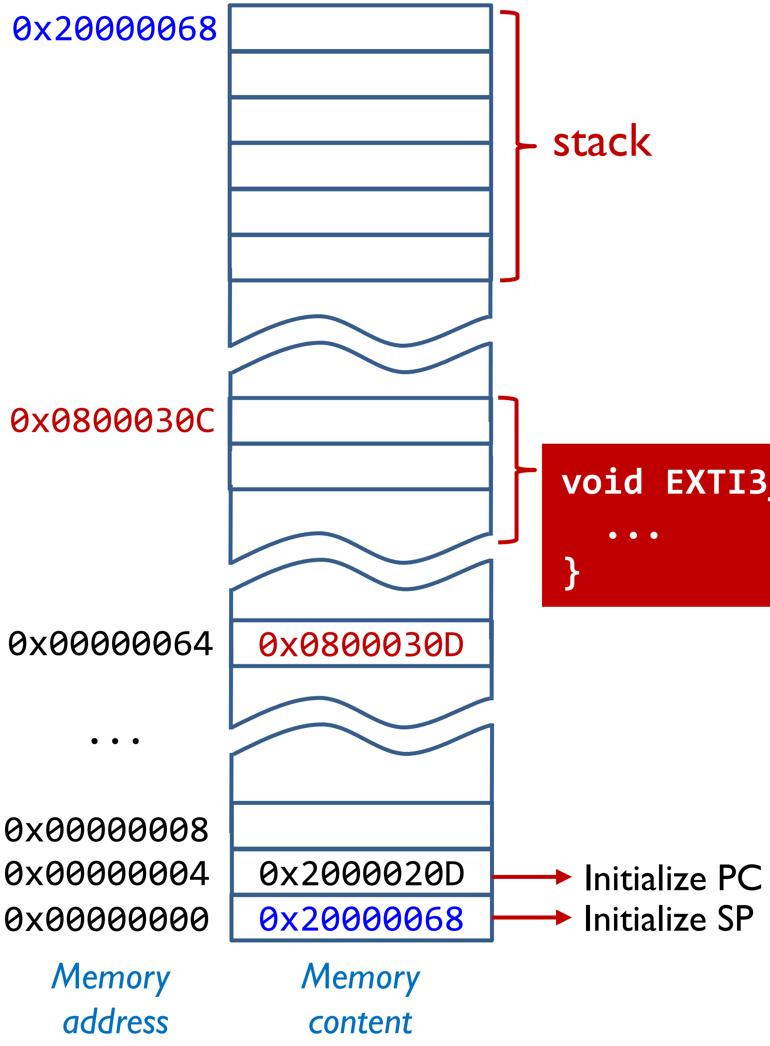
Full Descending Stack

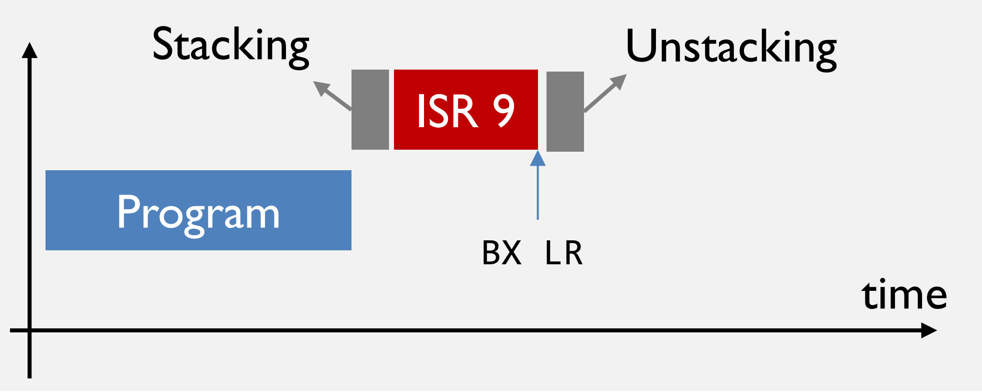
xxxxxxxx	SP + 0x20
xPSR	SP + 0x1C
PC(r15)	SP + 0x18
LR(r14)	SP + 0x14
r12	SP + 0x10
r3	SP + 0x0C
r2	SP + 0x08
r1	SP + 0x04
SP → r0	SP

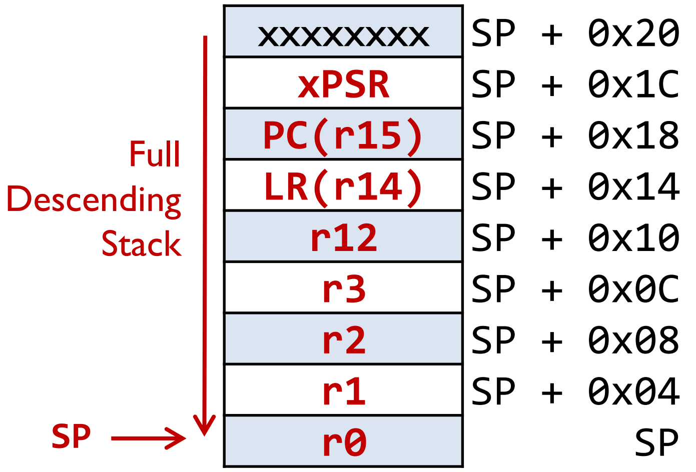
ISR 9 completes its execution by executing BX LR as its last instruction.

				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt







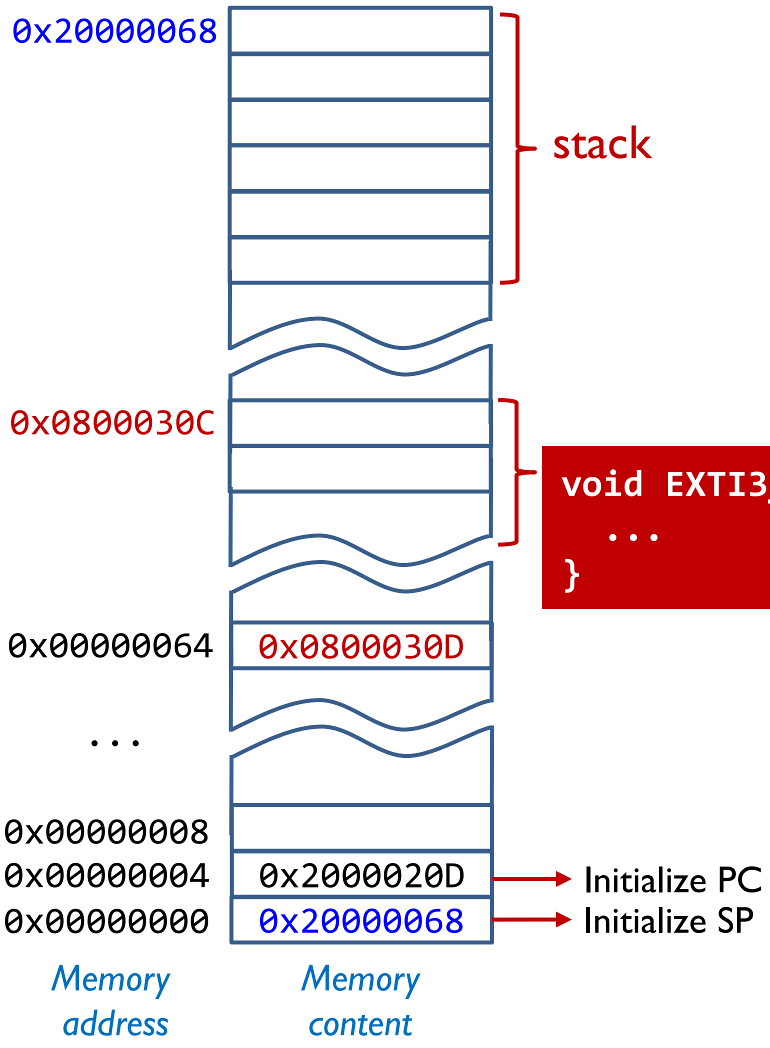
xxxxxxxx	SP + 0x20
xPSR	SP + 0x1C
PC(r15)	SP + 0x18
LR(r14)	SP + 0x14
r12	SP + 0x10
r3	SP + 0x0C
r2	SP + 0x08
r1	SP + 0x04
r0	SP

```
void EXTI3_IRQHandler{
  ...
}
```

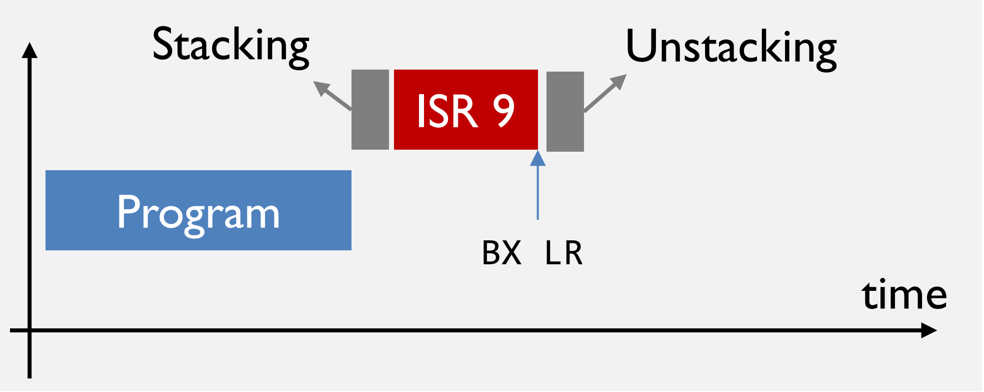
Instruction BX LR informs NVIC to perform the unstacking process. The active bit in Active Register is cleared. The unstacking process pops the values of the 8 registers, out of the stack. Therefore, the processor's state (running environment) is recovered, to the time instant immediately before ISR started.

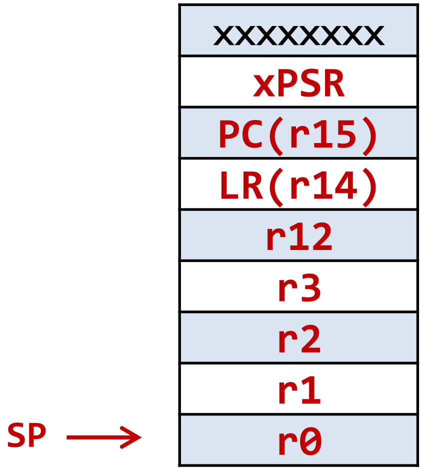
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
   ...
}
```

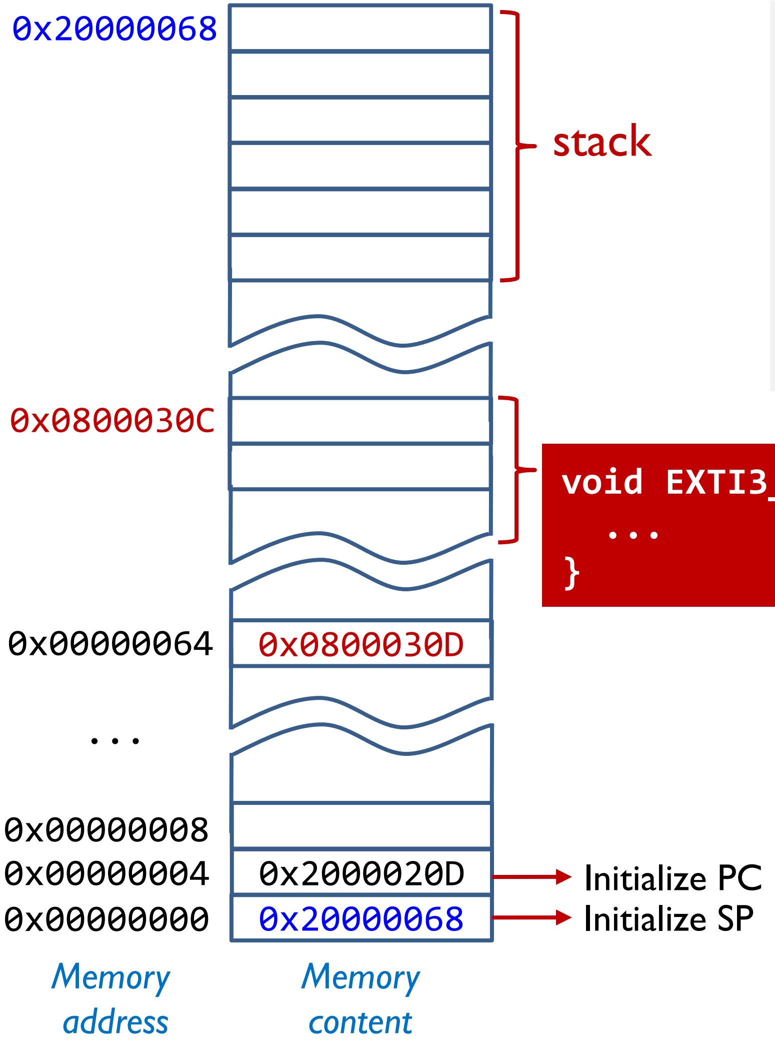




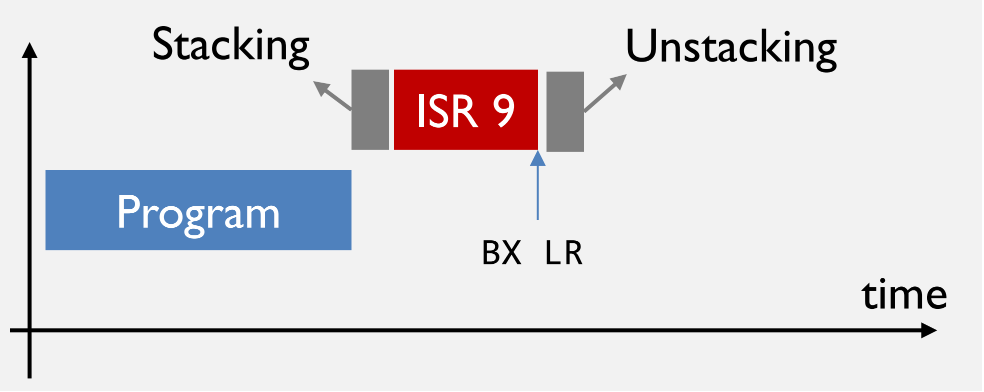
NVIC pops the 8 registers in the reverse order when they were pushed, thus recovering their original values.

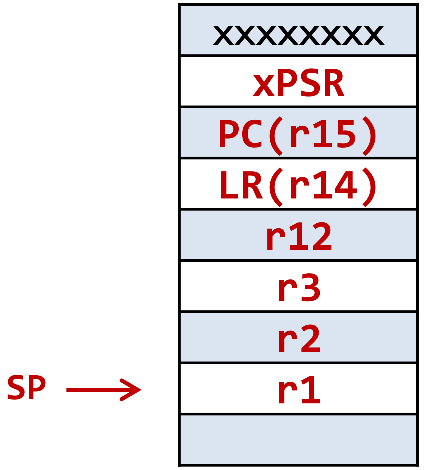
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



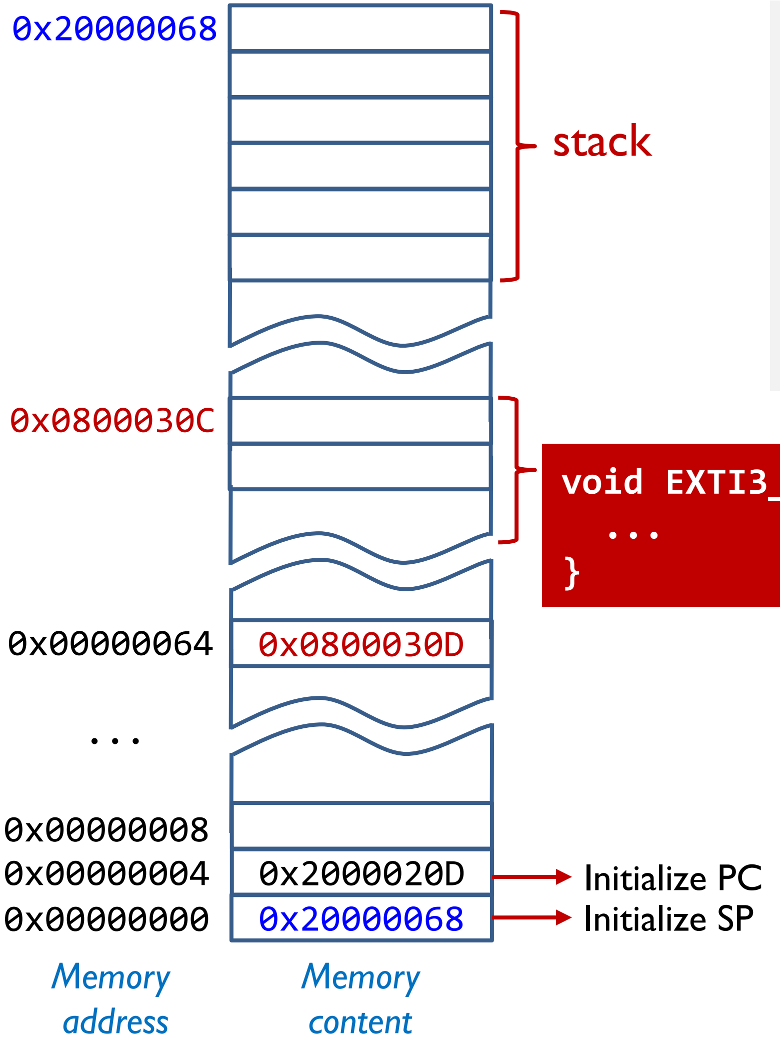
```
void EXTI3_IRQHandler{
  ...
}
```



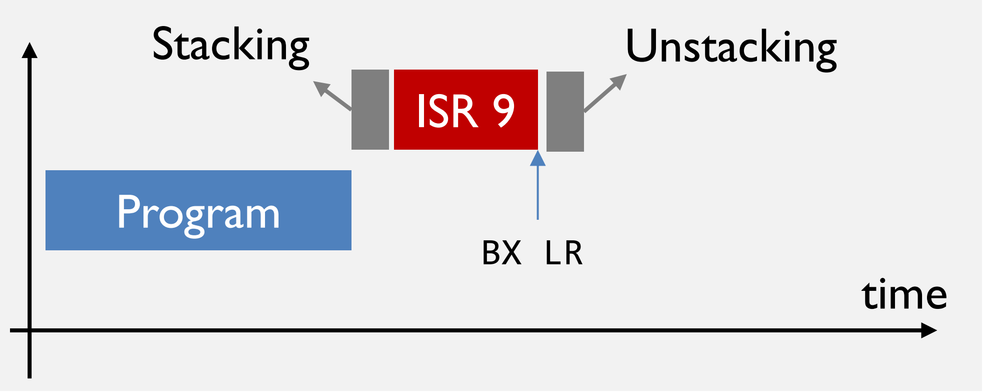


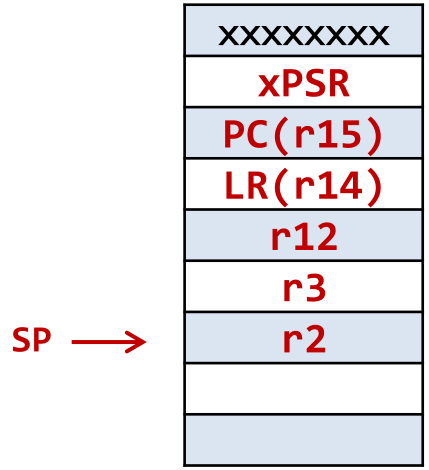
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



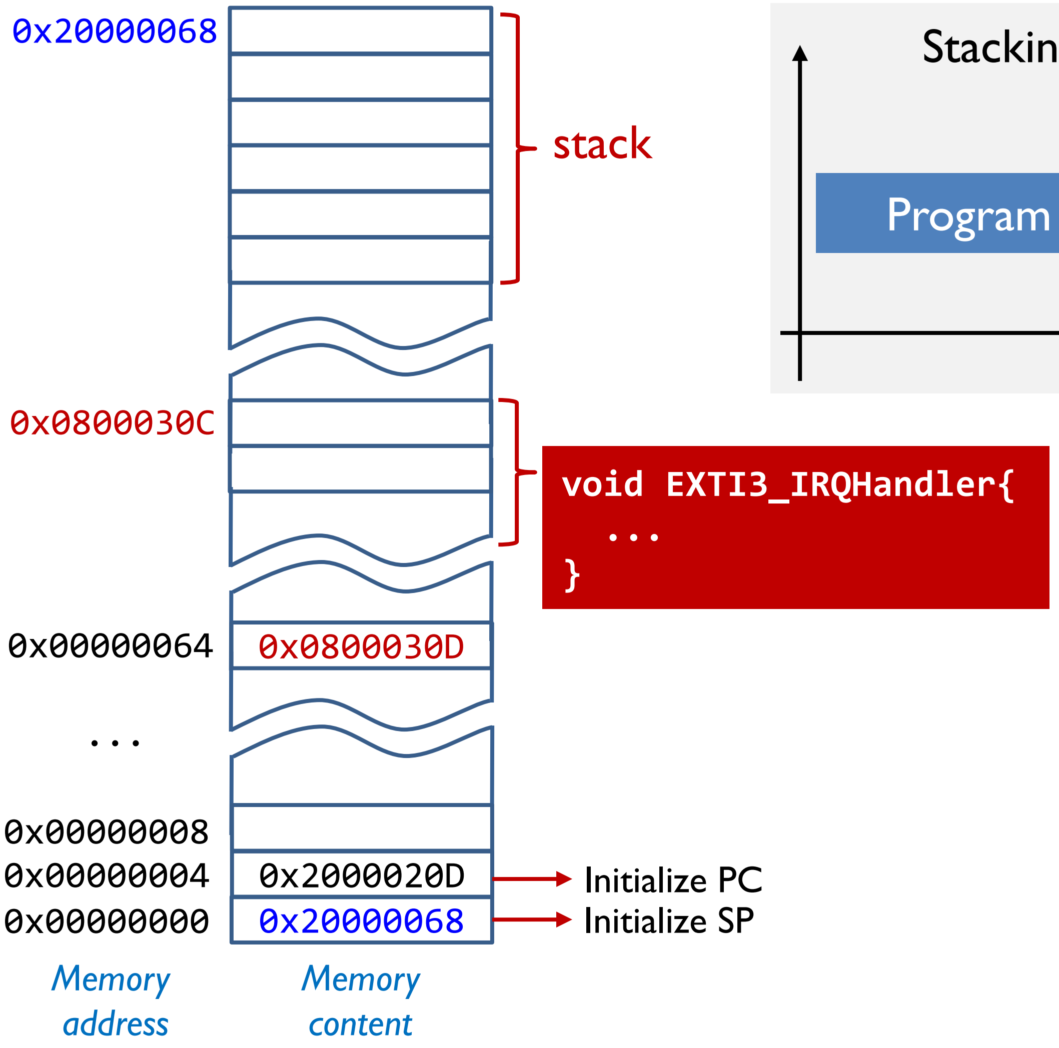
```
void EXTI3_IRQHandler{
  ...
}
```



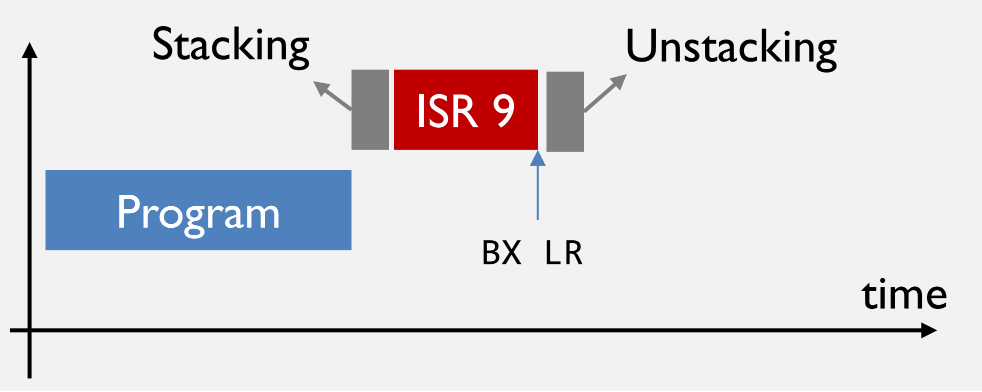


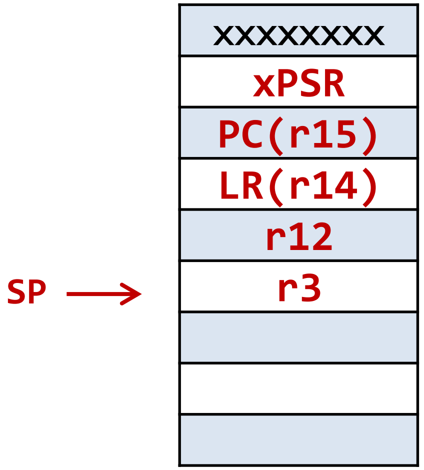
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



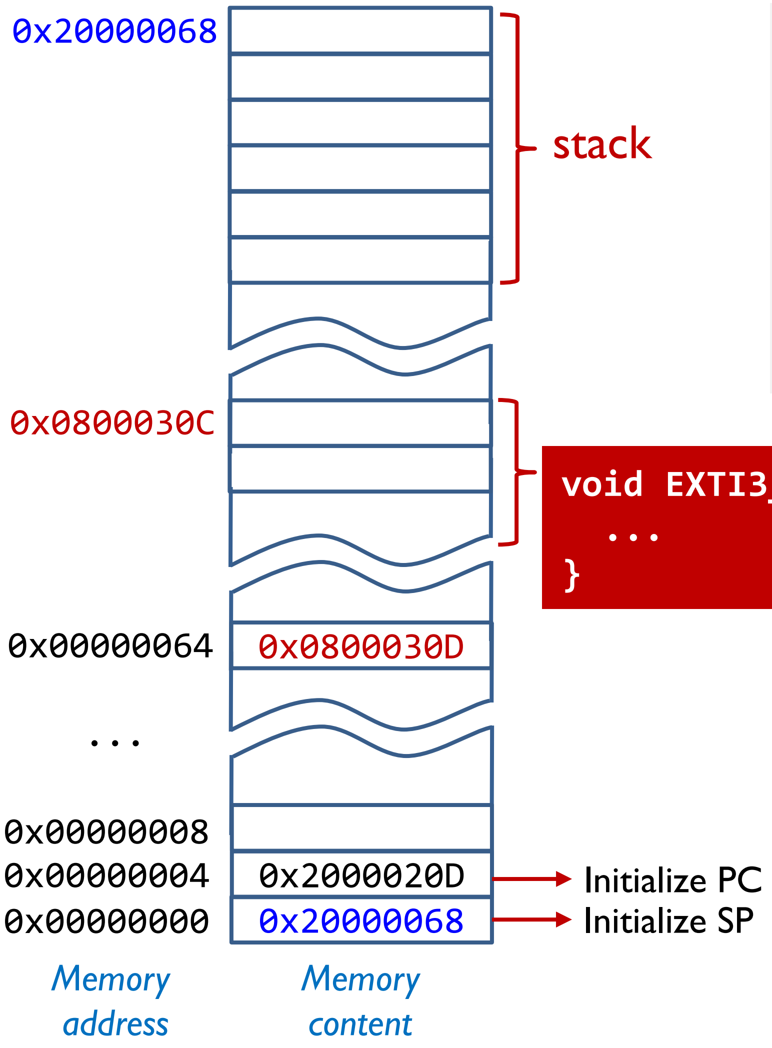
```
void EXTI3_IRQHandler{
  ...
}
```



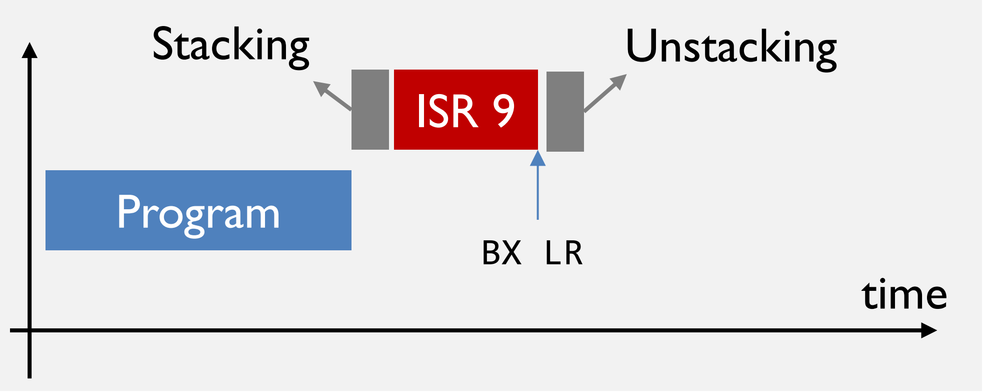


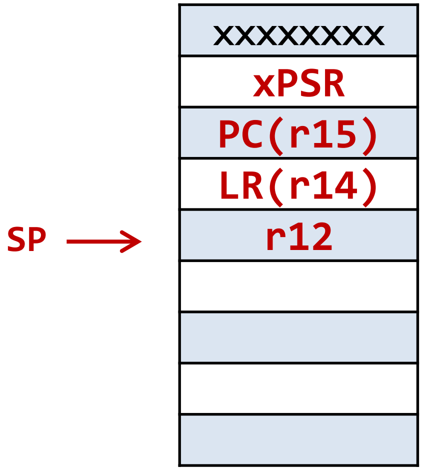
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



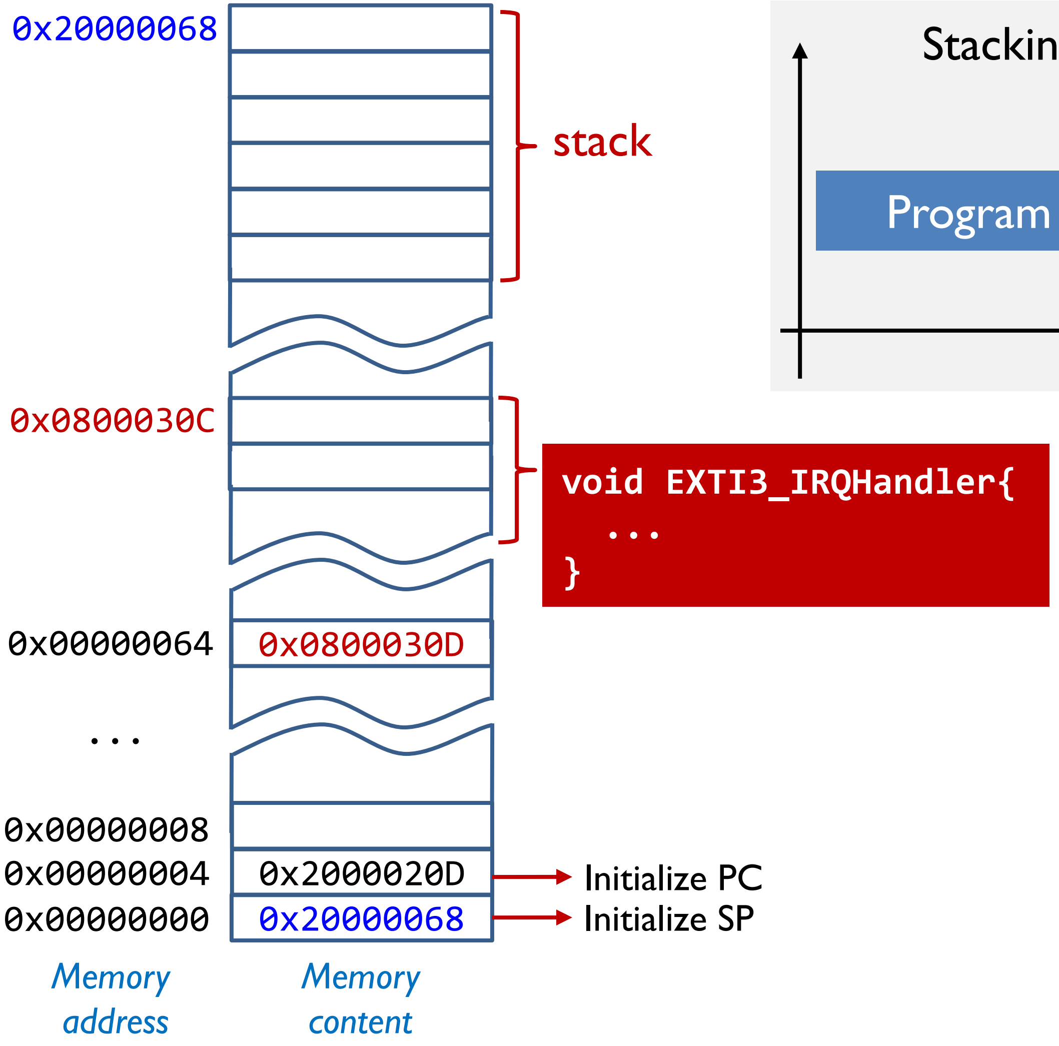
```
void EXTI3_IRQHandler{
  ...
}
```



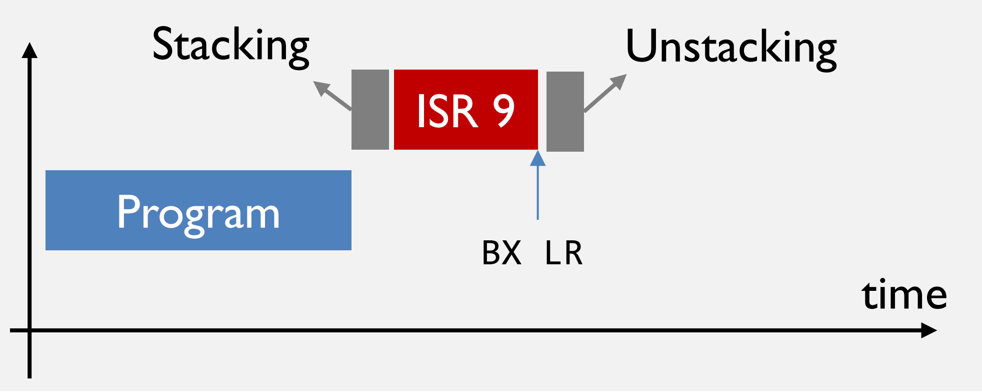


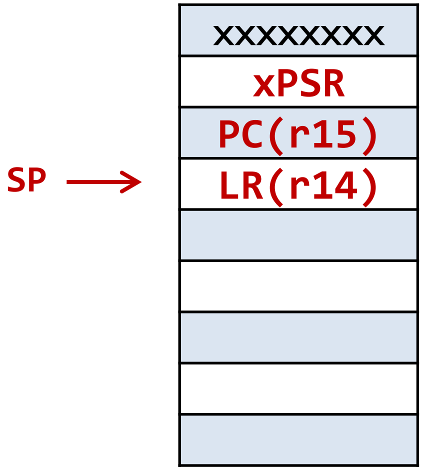
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



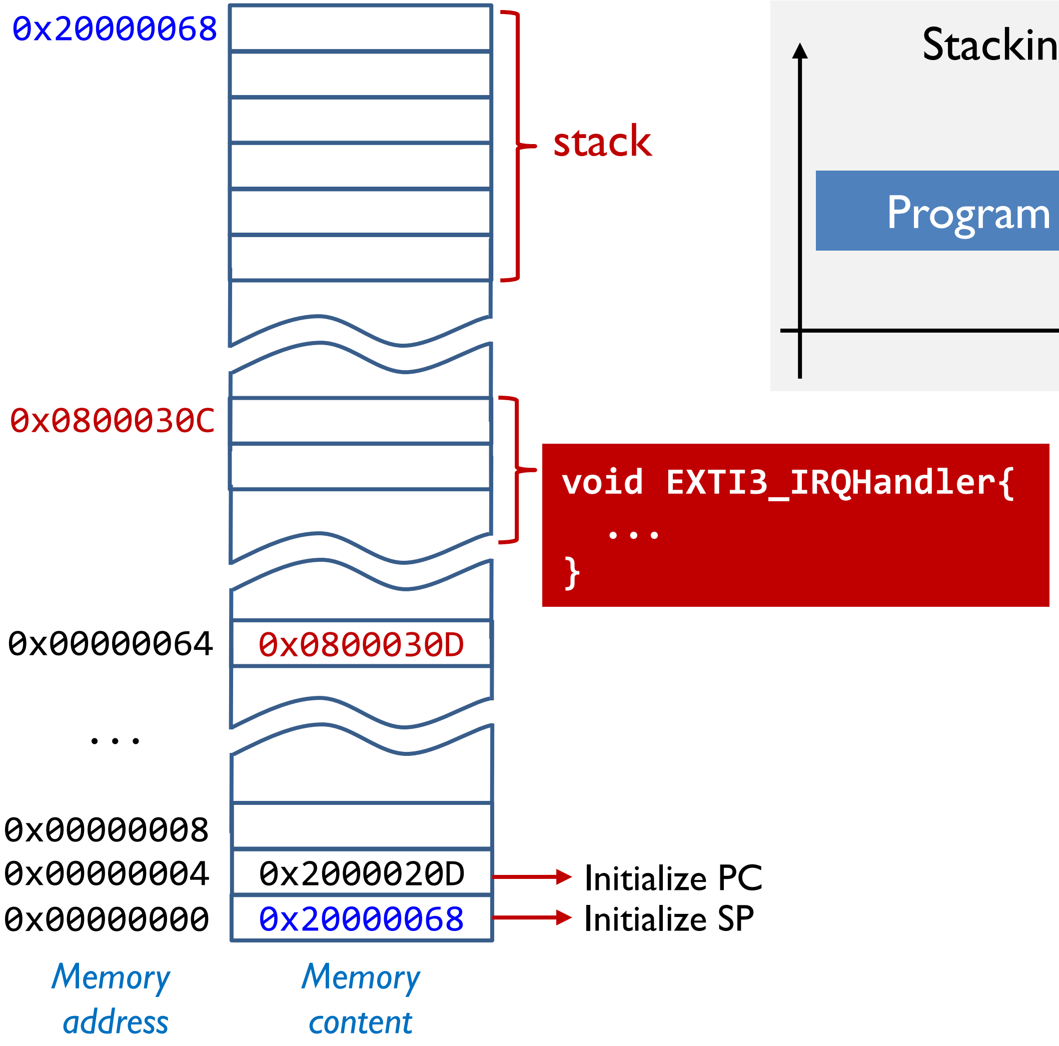
```
void EXTI3_IRQHandler{
  ...
}
```



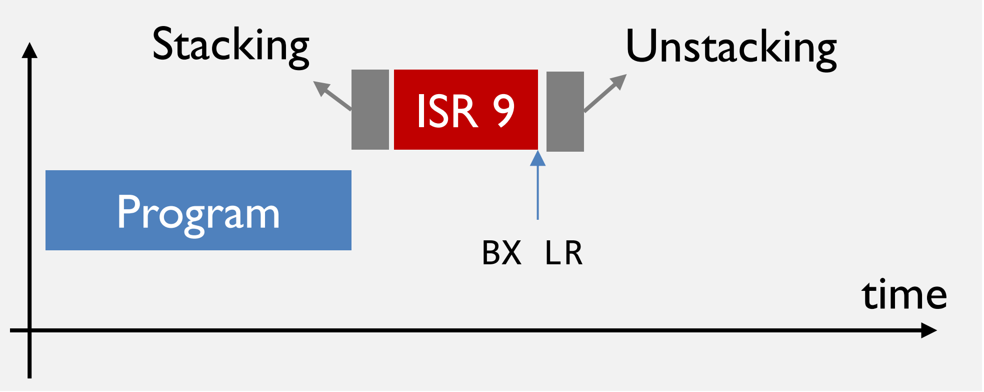


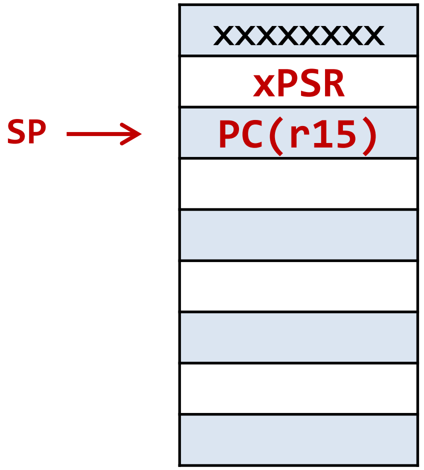
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



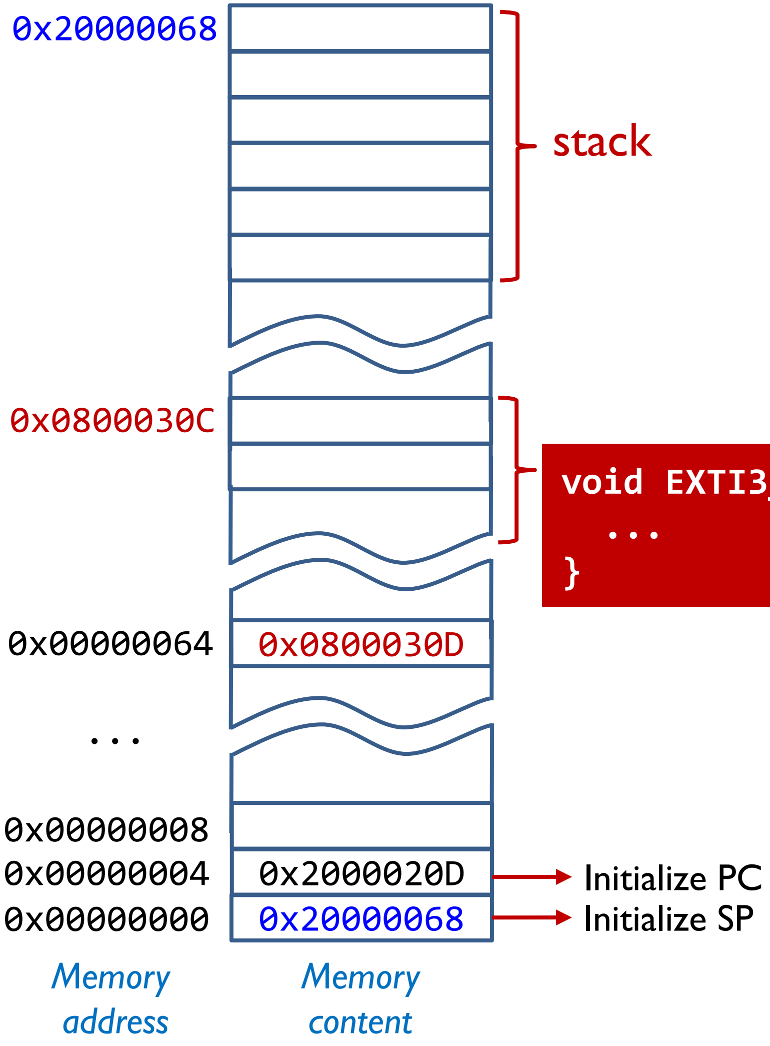
```
void EXTI3_IRQHandler{
  ...
}
```



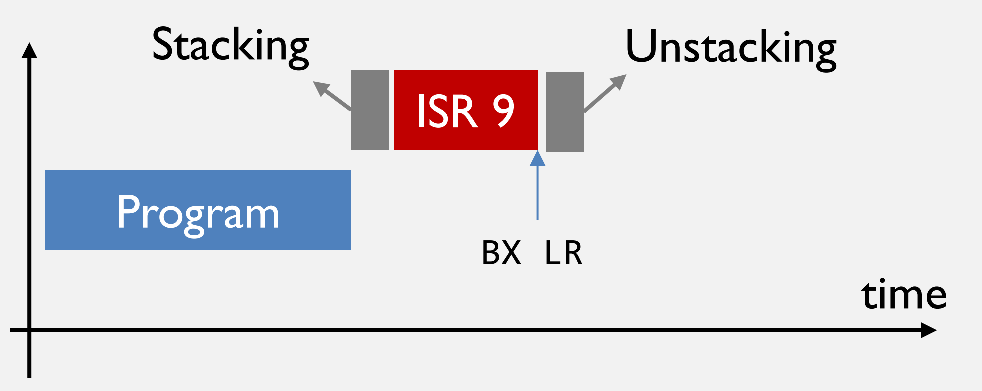


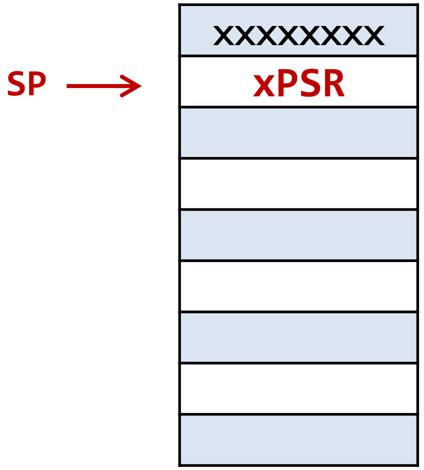
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



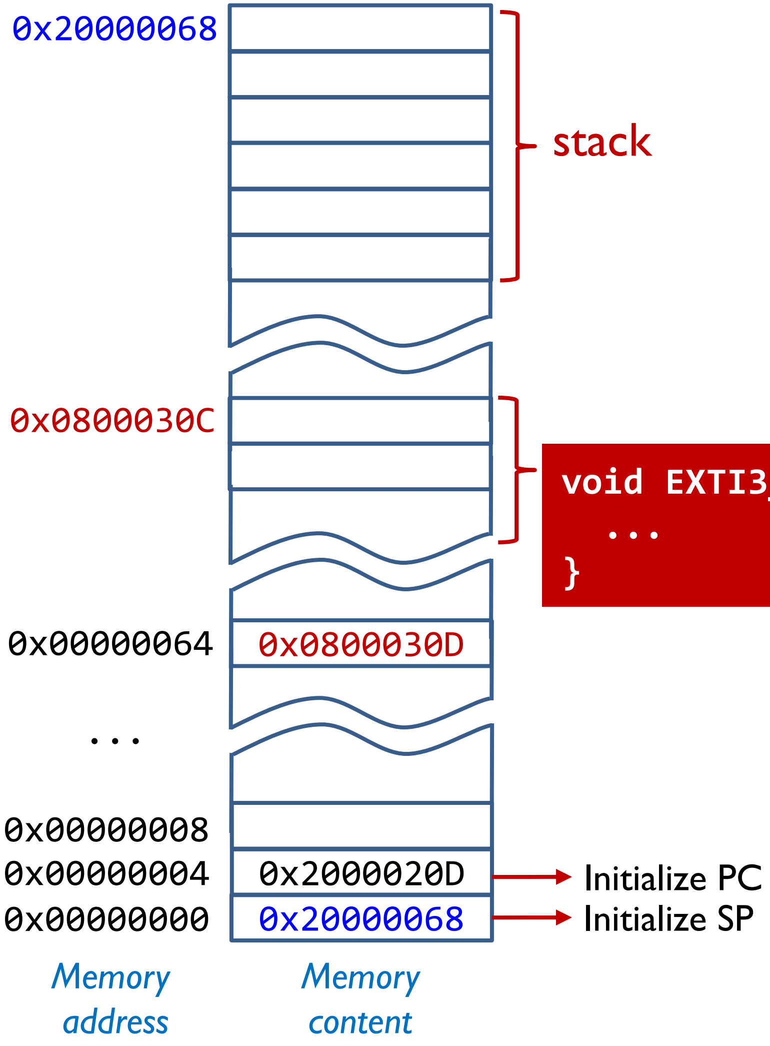
```
void EXTI3_IRQHandler{
  ...
}
```



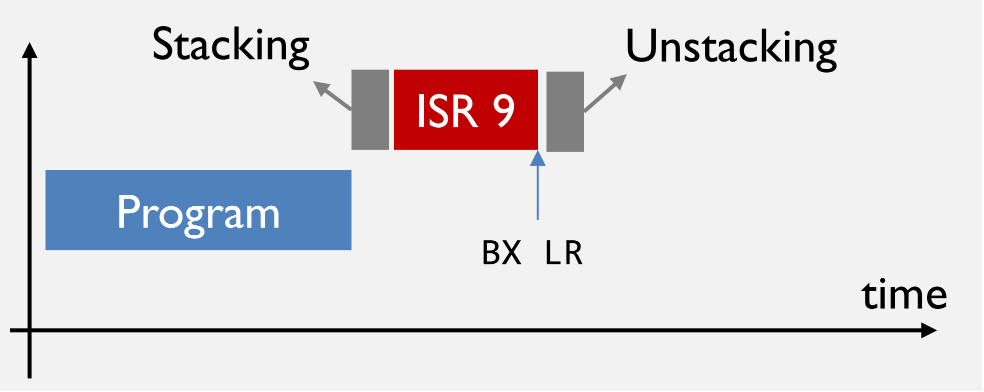


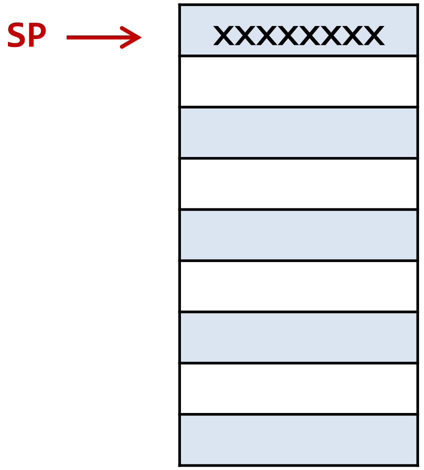
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt



```
void EXTI3_IRQHandler{
  ...
}
```

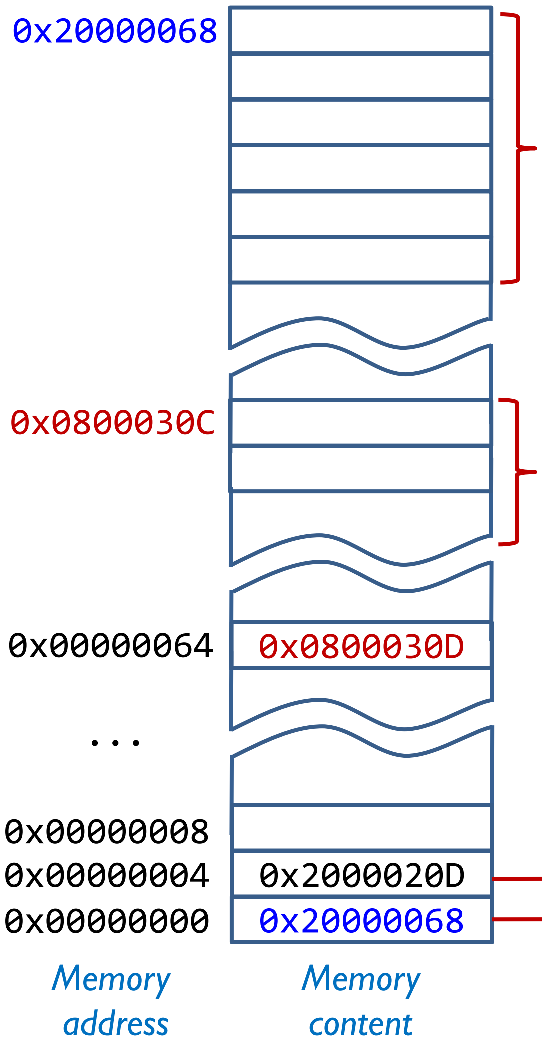




After unstacking completes, the running environment has been fully recovered from the stack. All registers have their original values, as if the interrupt has never happened.

				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Single Interrupt

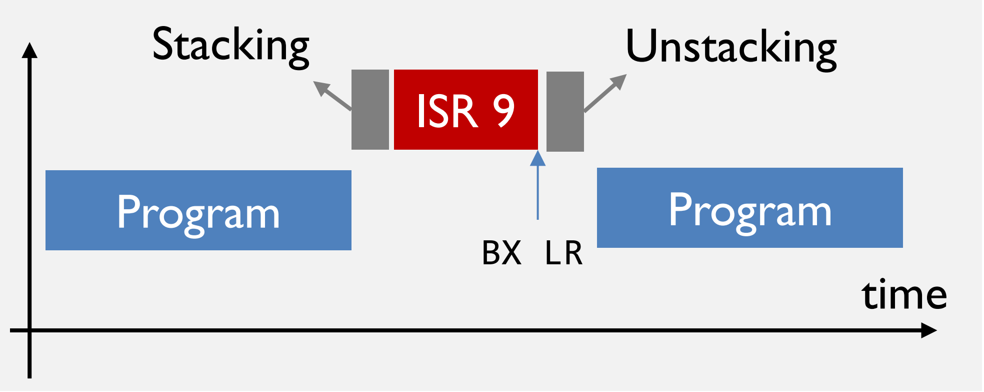


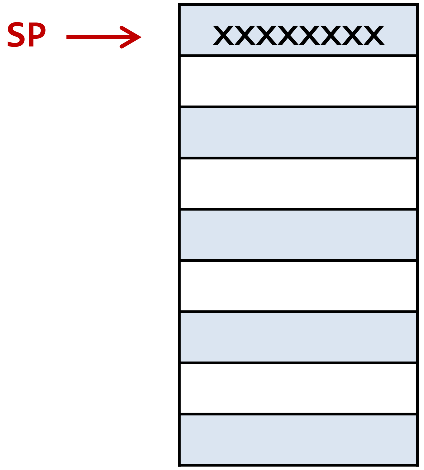
stack

```
void EXTI3_IRQHandler{
  ...
}
```

Initialize PC

Initialize SP

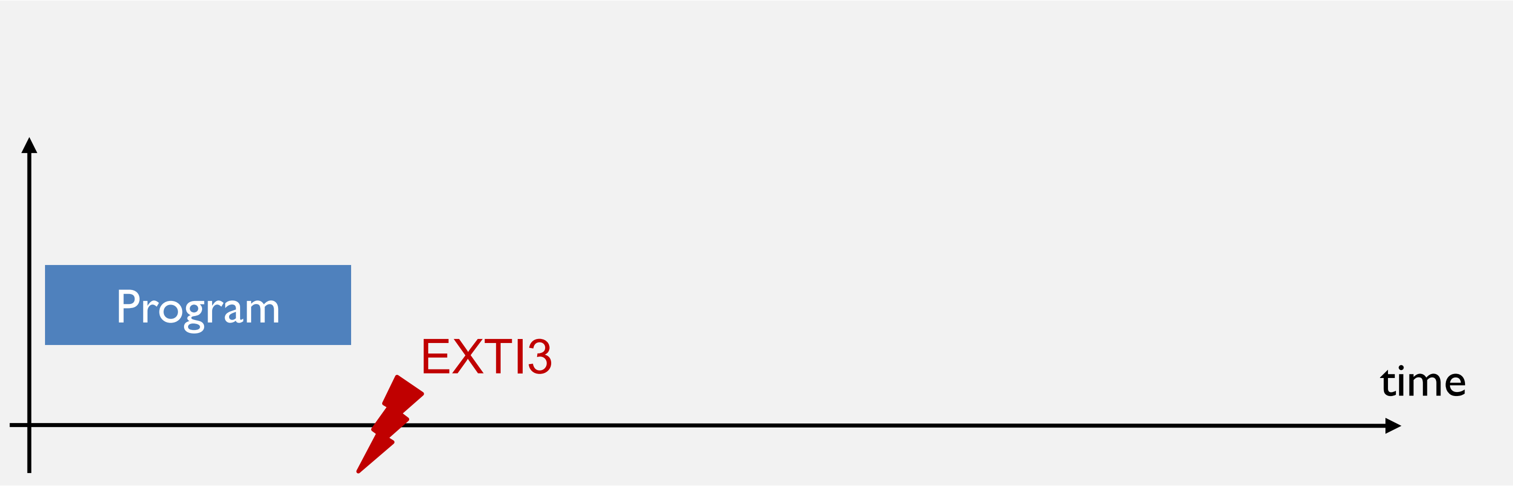




Afterwards, the processor continues to execute the user program, which was interrupted by EXTI3, ISR 9.

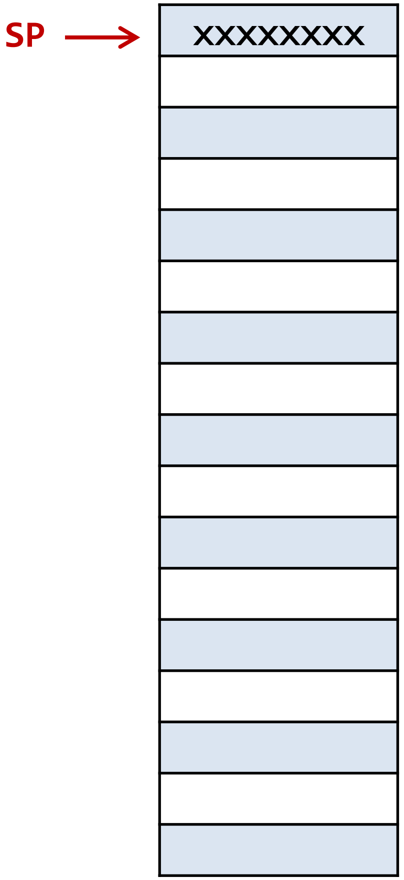
				EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	0	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	2	8

Nested Interrupts: Example of Preemption

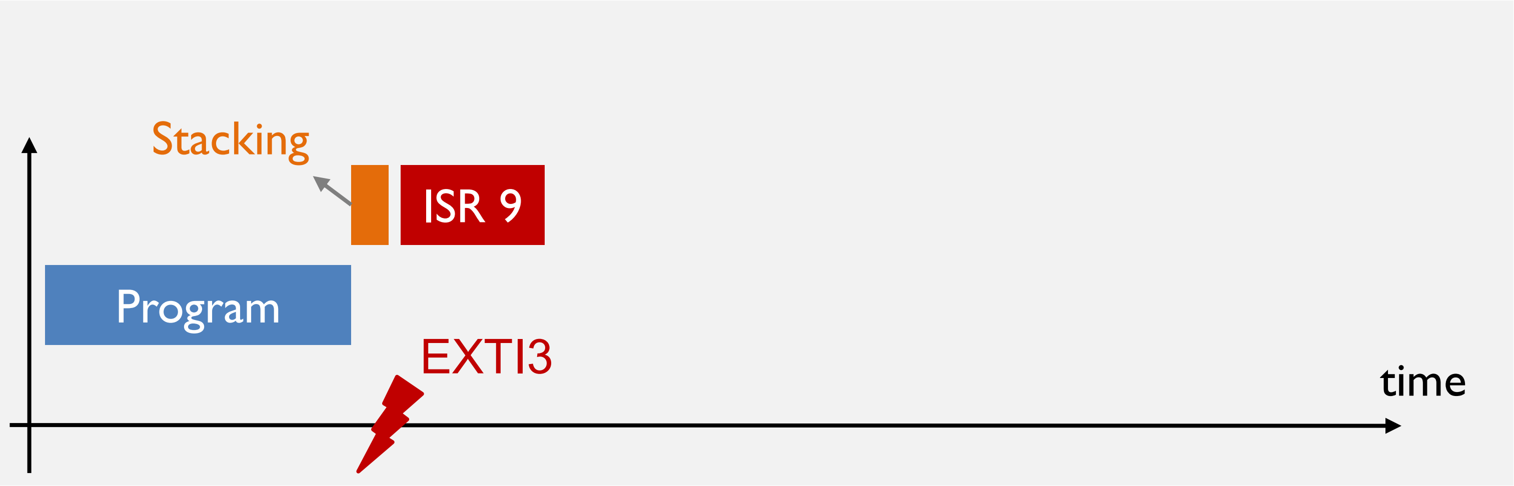


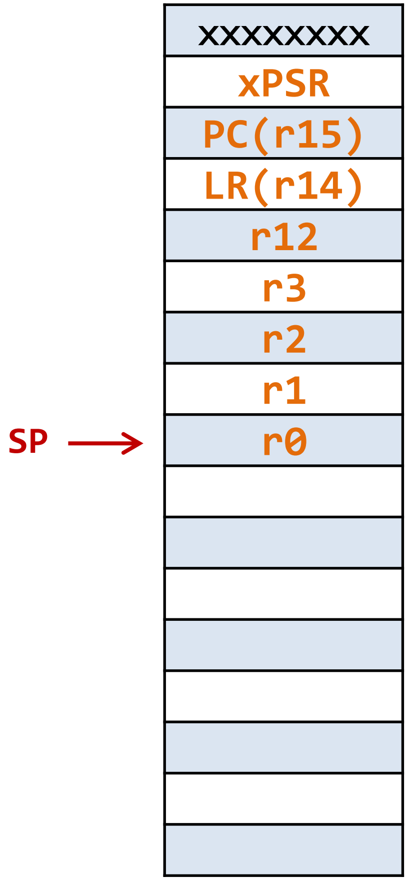
Suppose EXTI 3 arrives at this time instant.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3



Nested Interrupts:
Example of Preemption

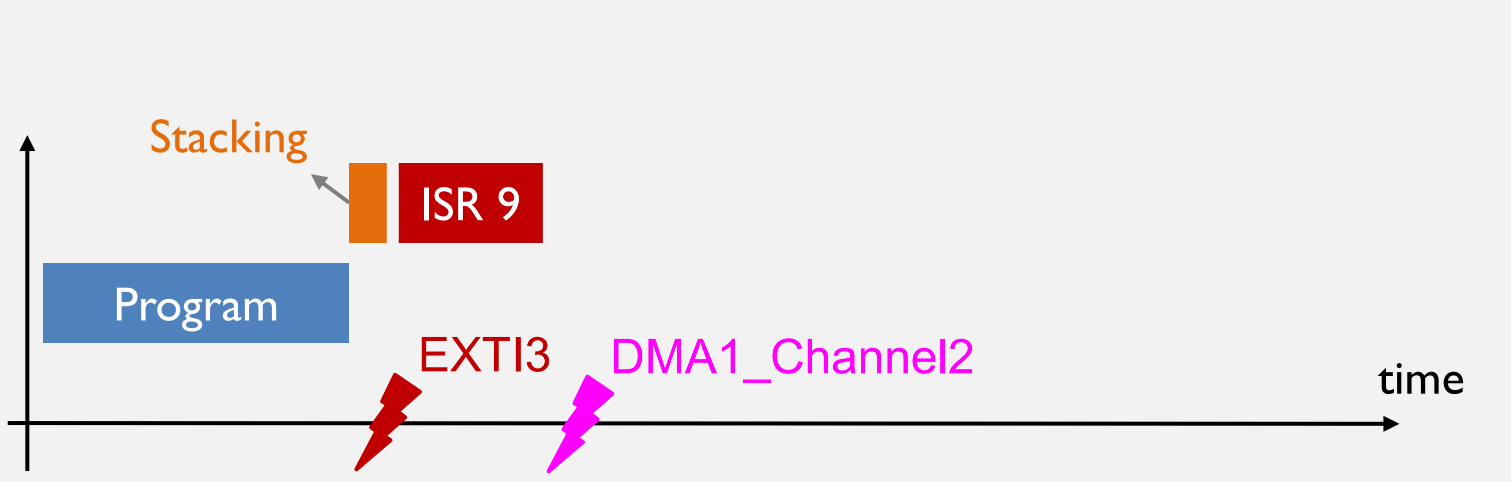




NVIC first performs stacking, and pushes 8 registers onto the stack. NVIC then forces the processor to execute ISR 9 for EXTI3.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3

Nested Interrupts:
Example of Preemption



Stacking

ISR 9

Program

EXTI3

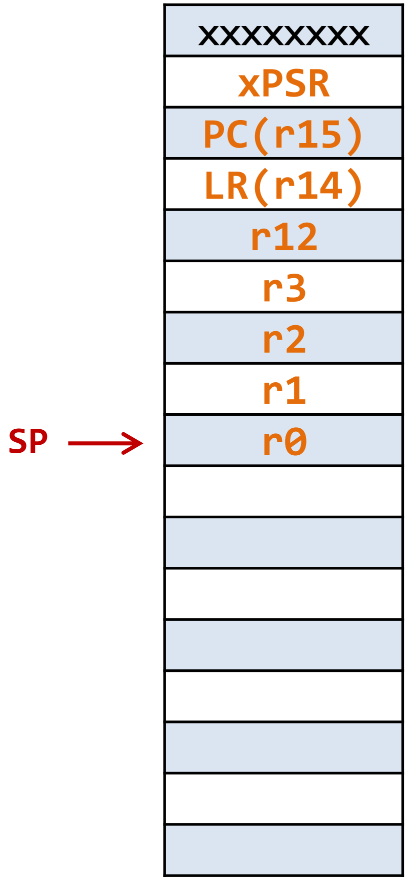
DMA1_Channel2

time

Suppose another interrupt (DMA 1 Channel 2) arrives, before ISR 9 completes. This new interrupt has higher urgency than the current interrupt being served, hence NVIC has to respond to the new coming interrupt.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	1	0	0	0	0
Priority Register	3	4	7	5	3

Lower priority value means higher urgency.



xxxxxxxx

xPSR

PC(r15)

LR(r14)

r12

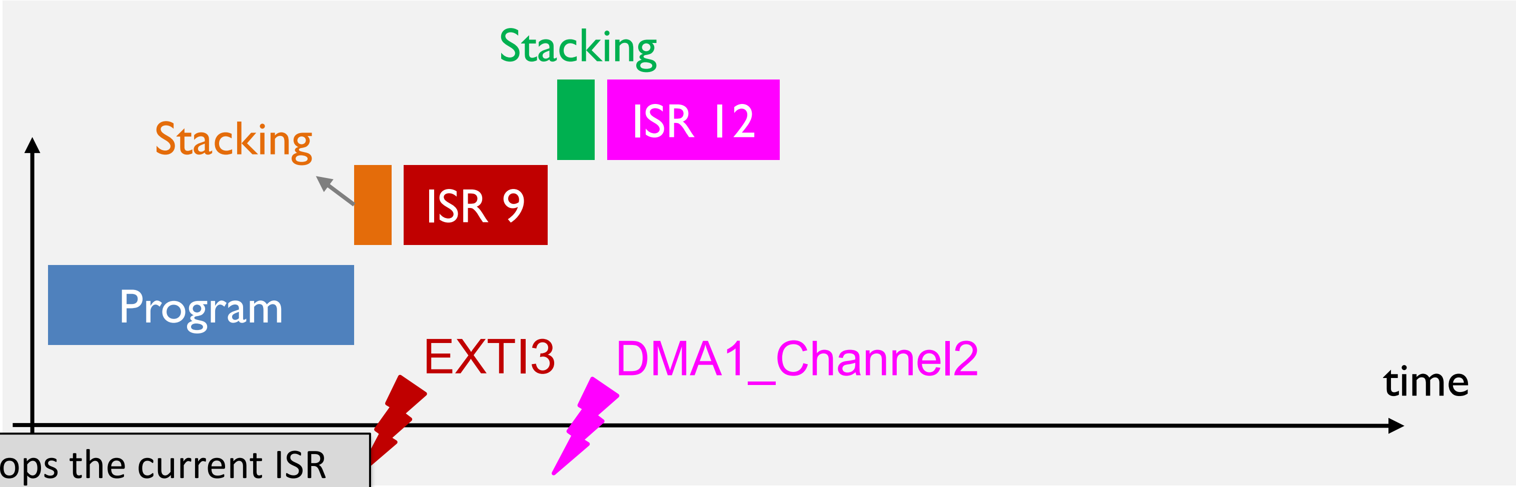
r3

r2

r1

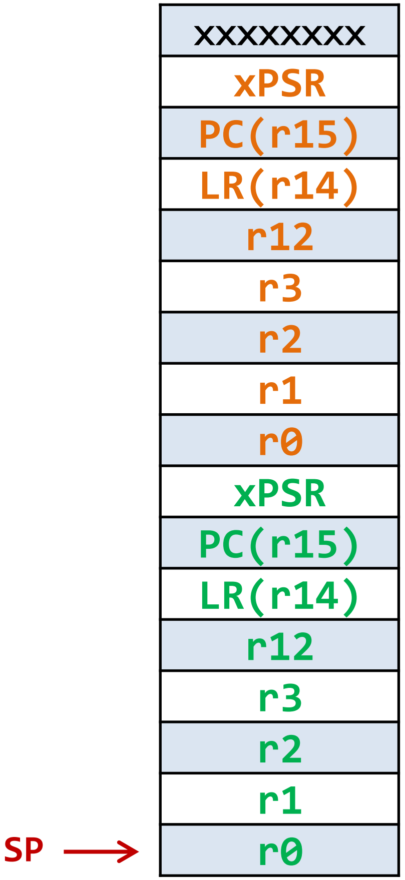
SP → r0

Nested Interrupts:
Example of Interrupt Preemption

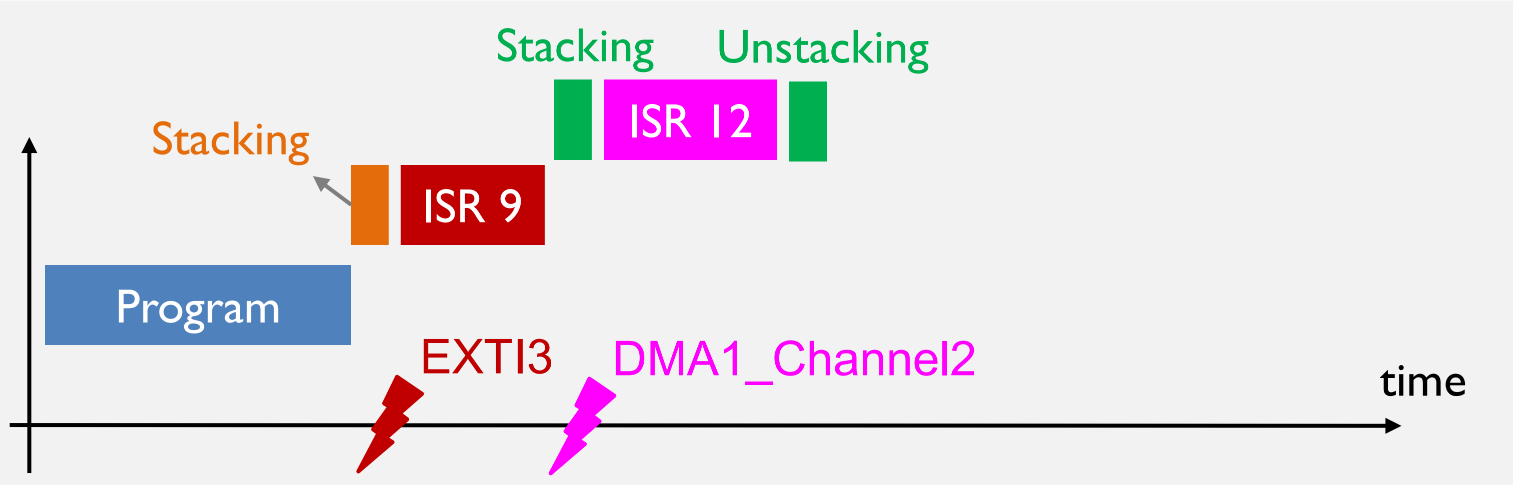


NVIC stops the current ISR 9 and performs another stacking by pushing another set of 8 registers, onto the stack. Note that these two sets of registers have different values. The first set holds register values, for the user program. The second set hold register values, for the interrupt service routine 9. After the stacking, NVIC starts to execute ISR 12 of the new coming interrupt.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	1	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3

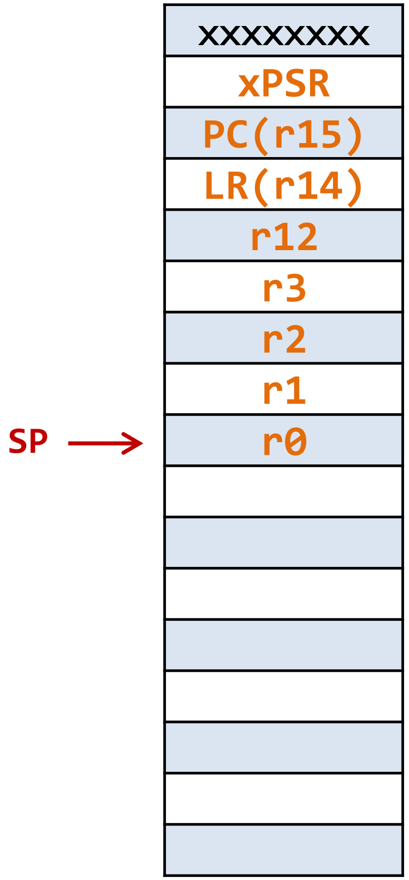


Nested Interrupts: Example of Preemption

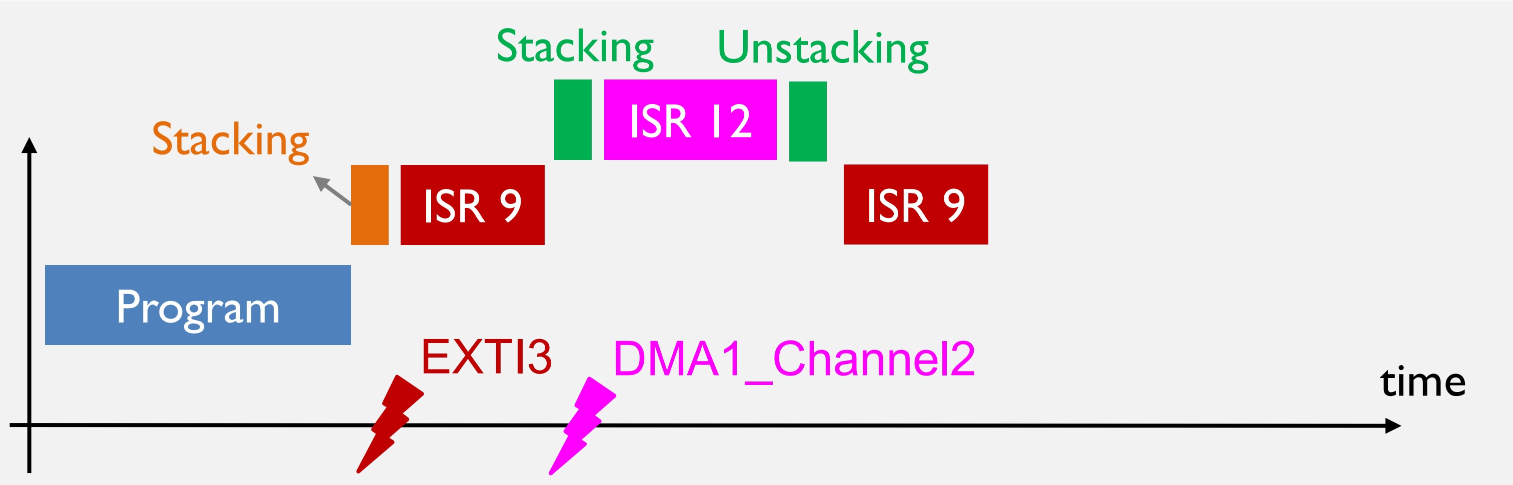


After ISR12 completes, NVIC performs unstacking, pops out eight registers from the stack, and recovers the running environment, for ISR 9.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3

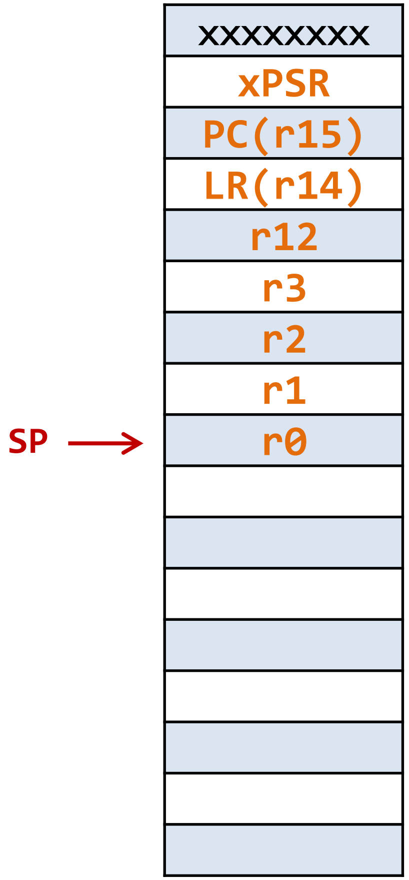


Nested Interrupts: Example of Preemption

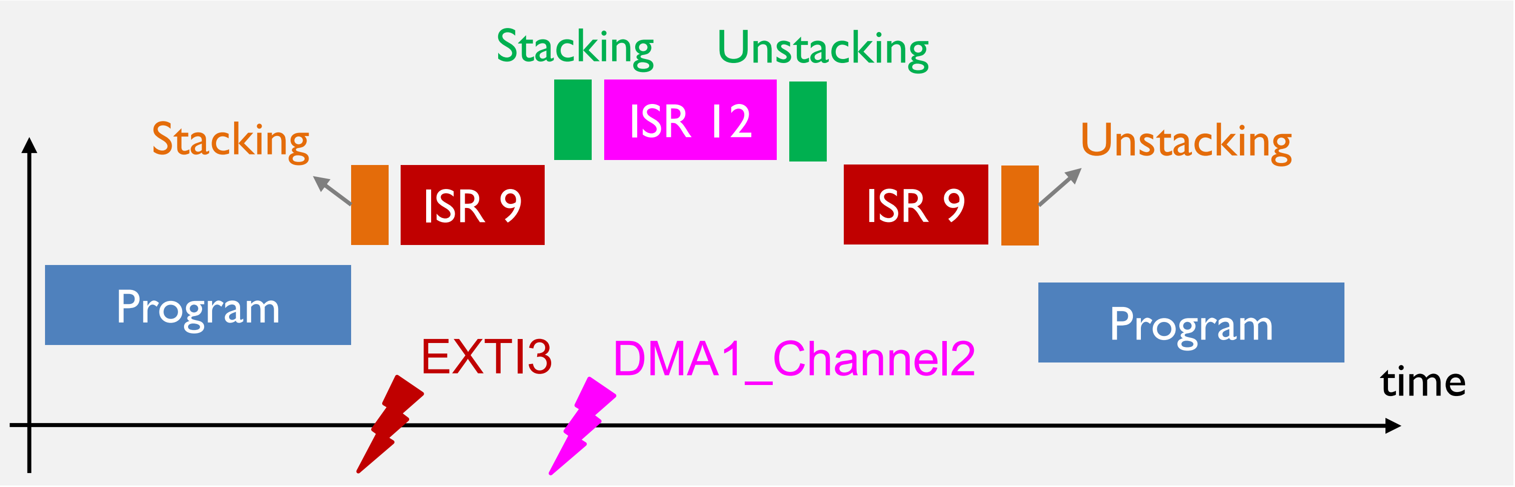


NVIC continues execution of ISR 9.

	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	1	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3

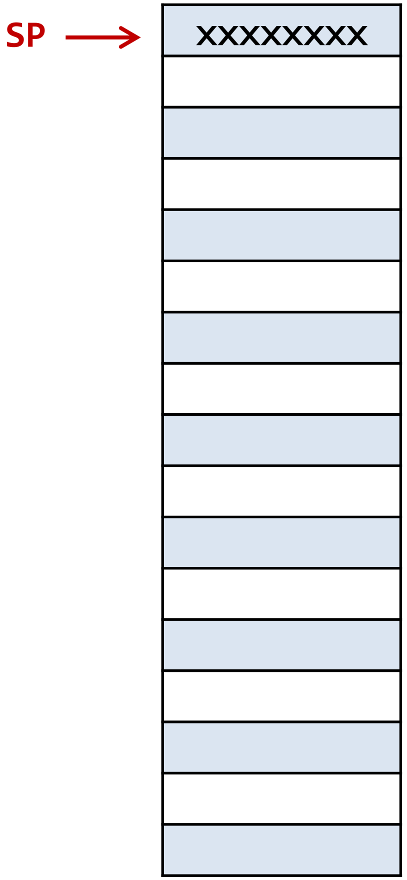


Nested Interrupts: Example of Preemption



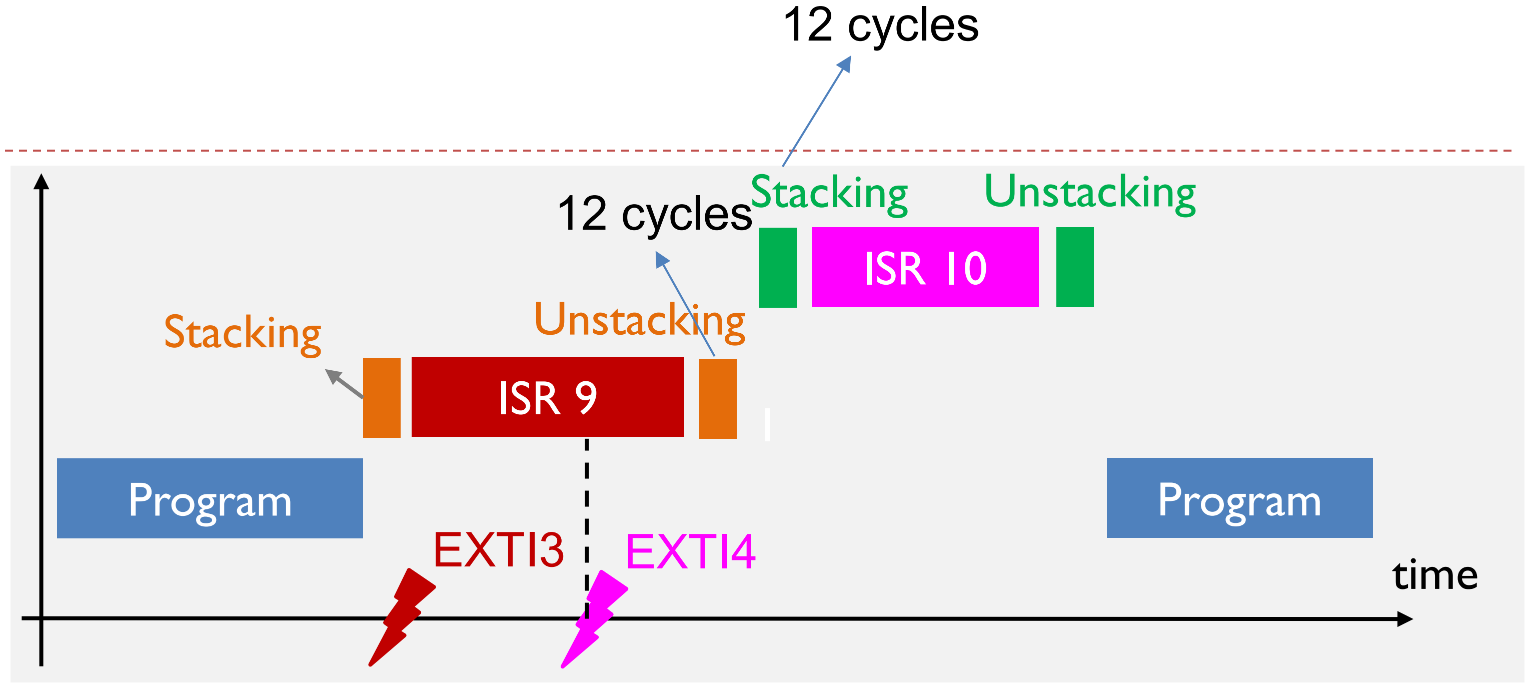
After ISR 9 completes, NVIC performs unstacking to recover the running environment of the user program. The user program then resumes its execution.

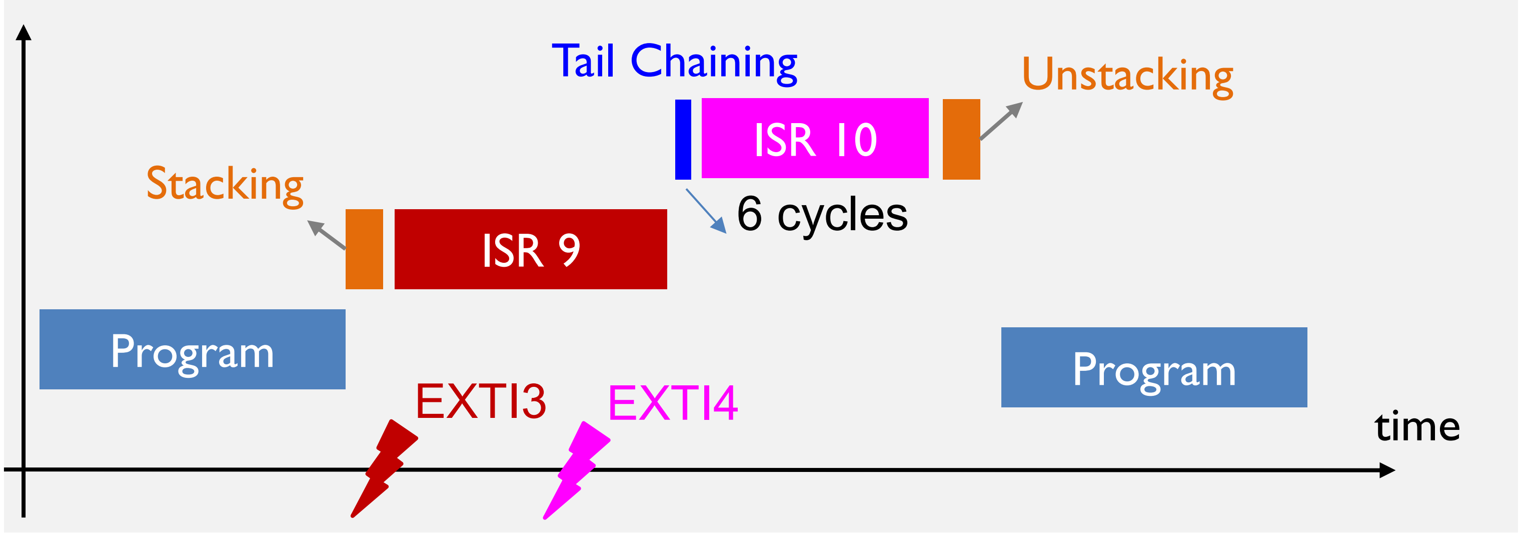
	DMA1_Channel2			EXTI3	
Interrupt Number	12	11	10	9	8
Enable Register	1	0	0	1	0
Active Register	0	0	0	0	0
Pending Register	0	0	0	0	0
Priority Register	3	4	7	5	3



Nested Interrupts: Tail Chaining

- EXTI3 → ISR 9
- EXTI4 → ISR 10
- Suppose EXTI4 has lower urgency than EXTI3.
  - EXTI4 has a higher numeric priority value than EXTI3.
  - If interrupt 4 EXTI4 arrives before the interrupt 3 EXTI3's handler completes, NVICC will continue the execution of the current ISR 9 for EXTI3. After it completes, unstacking and stacking are performed, before the new ISR 10 for EXTI4 starts.
- The middle unstacking and stacking are unnecessary in this example. Tail chaining is an optimization technique to reduce the interrupt latency.
  - Typically unstacking and stacking each takes 12 cycles.. However, tail chaining takes only 6 cycles.





References

- Lecture 9: Interrupts
  - https://www.youtube.com/watch?v=uFBNf7F3I60&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=9