



# The Cortex-M3 Embedded Systems





# Chapter 1, 2, and 3 in the reference book





## 8.1 Embedded System

### 8.1.1 Definition

An **embedded system** is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints.

Compare:

A general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs.

## 8.1.2 History

In the 1930–40s, computers were sometimes dedicated to a single task, but were far too large and expensive.

One of the first recognizably modern embedded systems was the **Apollo Guidance Computer**, developed by Charles Stark Draper at the MIT.

The first microprocessor, the **Intel 4004**, was designed for calculators and other small systems.

By the mid-1980s, most of the common previously external system components had been integrated into the same chip.

The integration of microcontrollers has increased the applications for which traditionally a computer would not.



## 8.1.3 Characteristics

1. Embedded systems are designed to do some specific task.  
Some also have real-time performance constraints that must be met; others may have low or no performance requirements.
2. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose.
3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips.



## 8.1.4 Application

Embedded systems span all aspects of modern life and there are many examples of their use.

- Telecommunications systems employ numerous embedded systems from telephone switches for the network to mobile phones at the end-user.
- Consumer electronics include personal digital assistants (PDAs), mp3 players, mobile phones and videogame consoles.





- Transportation systems from flight to automobiles increasingly use embedded systems.
- New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements.



- Various electric motors — brushless DC motors, induction motors and DC motors — are using electric/electronic motor controllers.

## 8.1.5 Processors in Embedded Systems

Embedded processors can be broken into two broad categories: ordinary **microprocessors** ( $\mu$ P) and **microcontrollers** ( $\mu$ C).

1. A fairly large number of basic CPU architectures are used.

Von Neumann as well as Harvard architectures.

RISC as well as non-RISC and VLIW.

2. Word lengths vary from 4-bit to 64-bits and beyond (mainly in DSP processors).

3. Most architectures come in a large number of different variants and shapes.



## ARM

ARM, a company was known as the ***Advanced RISC Machine***, and before that as the ***Acorn RISC Machine***.



As of 2007, about 98 percent of the more than one billion mobile phones sold each year use at least one ARM processor.

As of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors.

ARM processors are used extensively in consumer electronics, including PDAs, mobile phones, digital media and music players.

**ARM's business has always been to sell the Intellectual Property of microprocessor cores.**

The most successful implementation has been the ARM7TDMI with hundreds of millions sold.

Companies that are current or former ARM licensees include:

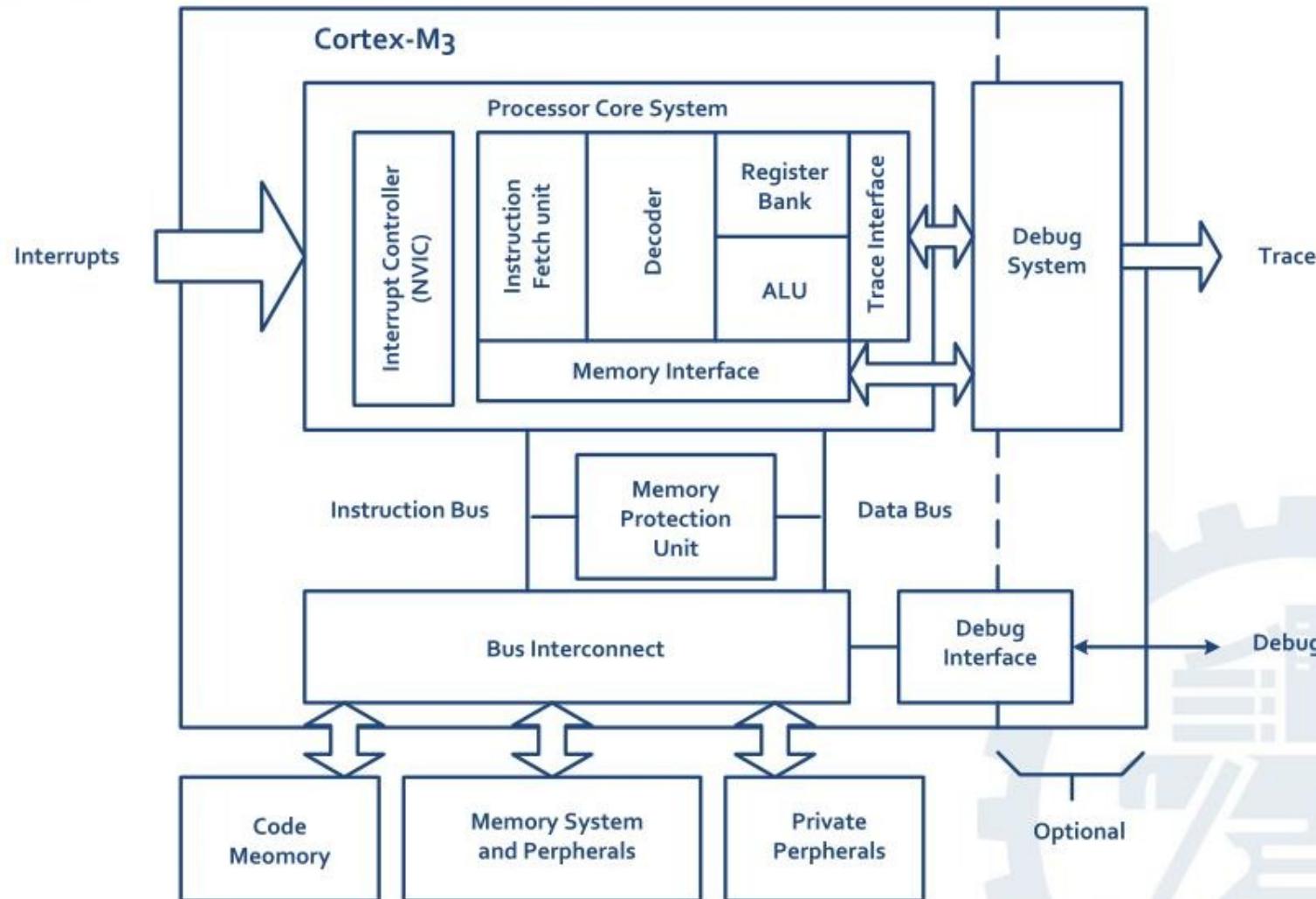
Alcatel-Lucent, Apple Inc., Atmel, Broadcom, Freescale, Intel, Samsung, TI, ect.



Prominent examples of ARM Holdings ARM processor families include the ARM7, ARM9, ARM11 and Cortex.



## 8.2 Overview of Cortex-M3



A Simplified View of the Cortex-M3



1. **32-bit** microprocessor: 32-bit data path, 32-bit register bank, 32-bit memory interfaces.
2. **Harvard architecture**: A separate instruction bus and data bus. This allows instructions and data accesses to take place at the same time.
3. Memory space: **4GB**.
4. Registers: Registers (R0 to R15) and special registers.
5. **Operation Modes**: Thread mode and handler mode; Privileged level and user level.
6. Interrupts and Exceptions: The built-in **Nested Vectored Interrupt Controller**; Support 11 system exceptions plus 240 external IRQs.

- 
7. The Bus Interface: Several bus interfaces allow the Cortex-M3 to carry instruction fetches and data accesses at the same time.
  8. MPU: An optional Memory Protection Unit allows access rules to be set up for privileged access and user program access.
  9. **The Instruction Set:** Thumb-2 instruction set: allows 32-bit instructions and 16-bit instructions to be used together; no ARM instructions allowed
  10. Fixed internal debugging components: Provide debugging operation supports and features such as breakpoints, single step.



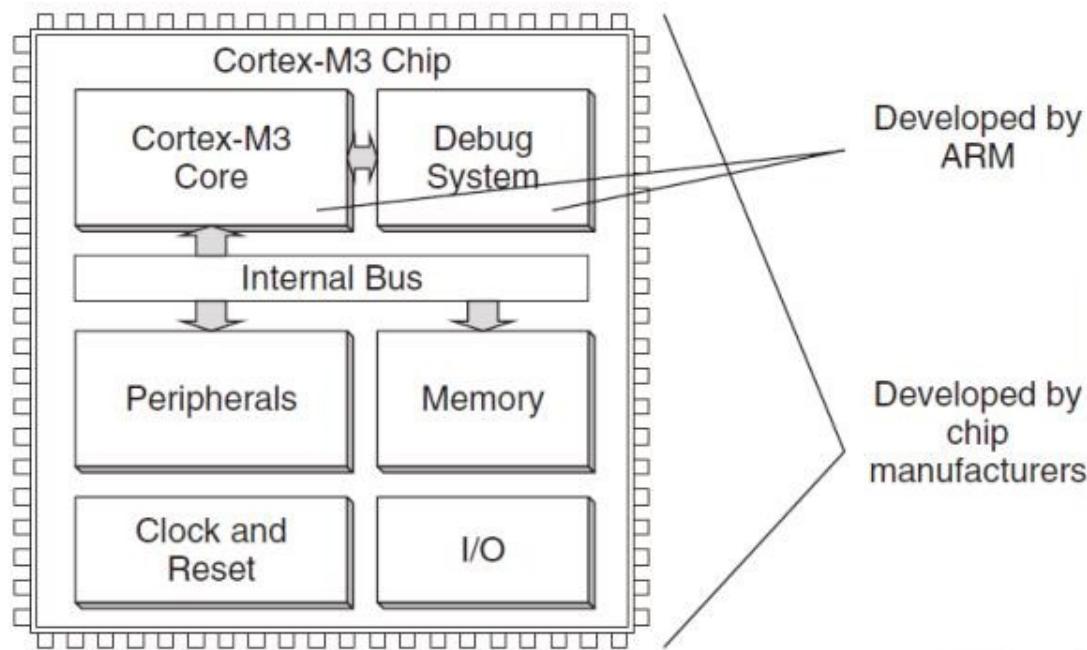
Primarily designed to target the 32-bit microcontroller market

- ④ **Greater performance efficiency**, allowing more work to be done without increasing the frequency or power requirements
- ④ **Low power consumption**, enabling longer battery life
- ④ **Enhanced determinism**, guaranteeing that critical tasks and interrupts are serviced in a known number of cycles
- ④ **Improved code density**, ensuring that code fits in even the smallest memory
- ④ **Ease of use**, providing easier programmability and debugging
- ④ **Lower-cost solutions**, reducing 32-bit-based system costs at less than US\$1 for the first time
- ④ **Wide choice of development tools**, from low-cost or free compilers to full-featured development suites

The Cortex-M3 processor is the central processing unit (CPU) of a microcontroller chip

In addition, a number of other components are required for the whole Cortex-M3 processor-based microcontroller

Cortex-M3 processor based chips from different manufacturers will have different memory sizes, types, peripherals, and features





Previous ARM processors support both ARM state and Thumb state

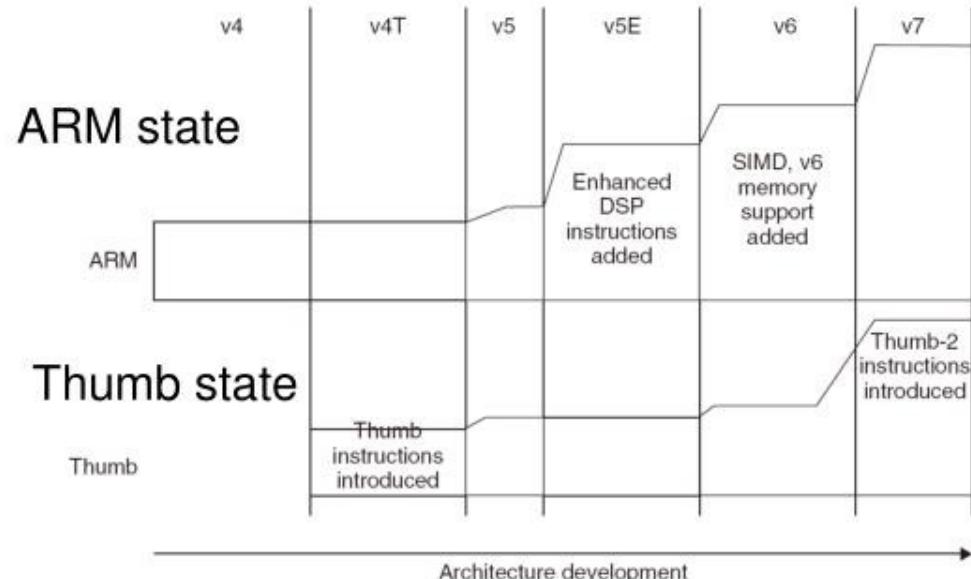


Figure 1.3 Instruction Set Enhancement

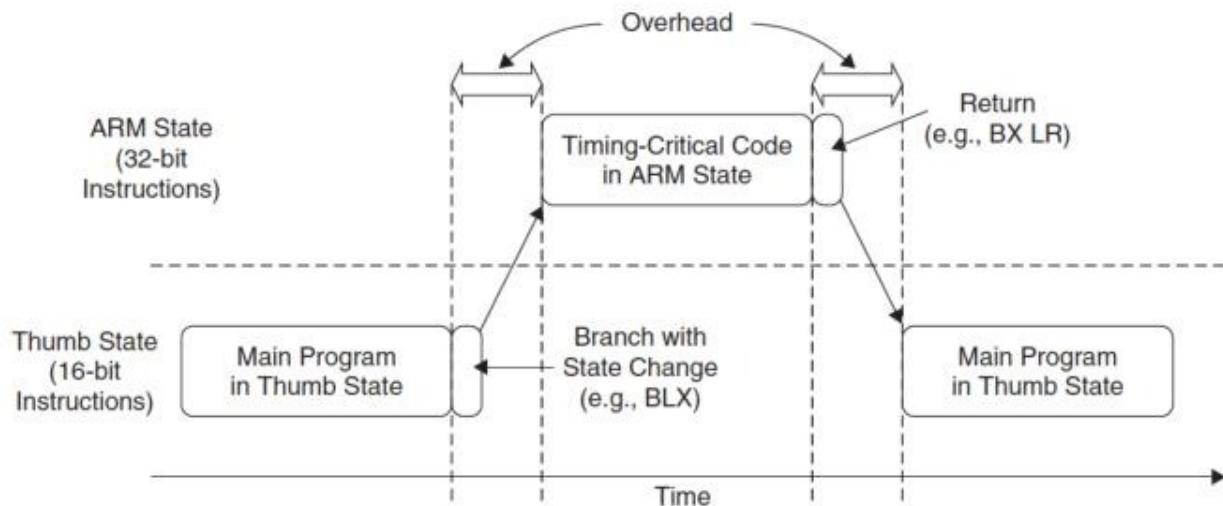
ARM instructions that are 32-bit

Thumb instructions that are 16-bit



- ④ In the ARM state, the instructions are 32-bit and can execute all supported instructions with very **high performance**
- ④ In the Thumb state, the instructions are 16-bit, so there is a much **higher instruction code density**

**Can we combine them to achieve the best of both worlds?**

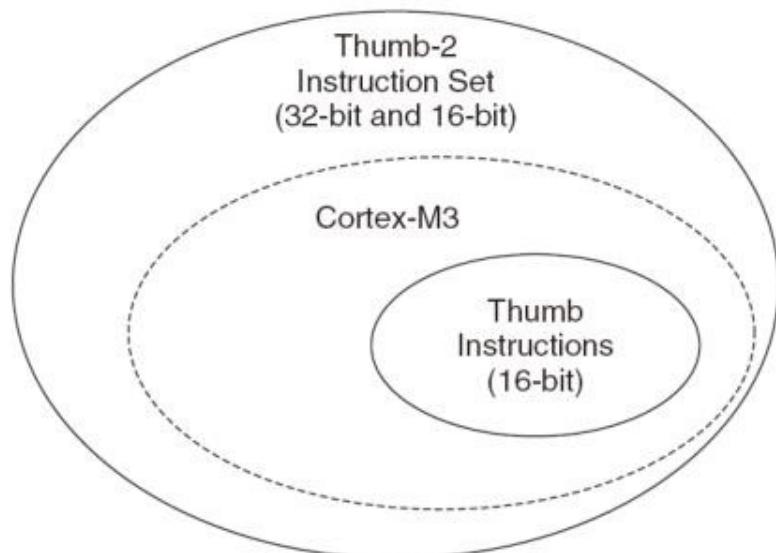


Problems:

- 1) overhead of state switches
- 2) complex software development



- ④ Cortex-M3 supports only the Thumb-2 (and traditional Thumb) instruction set

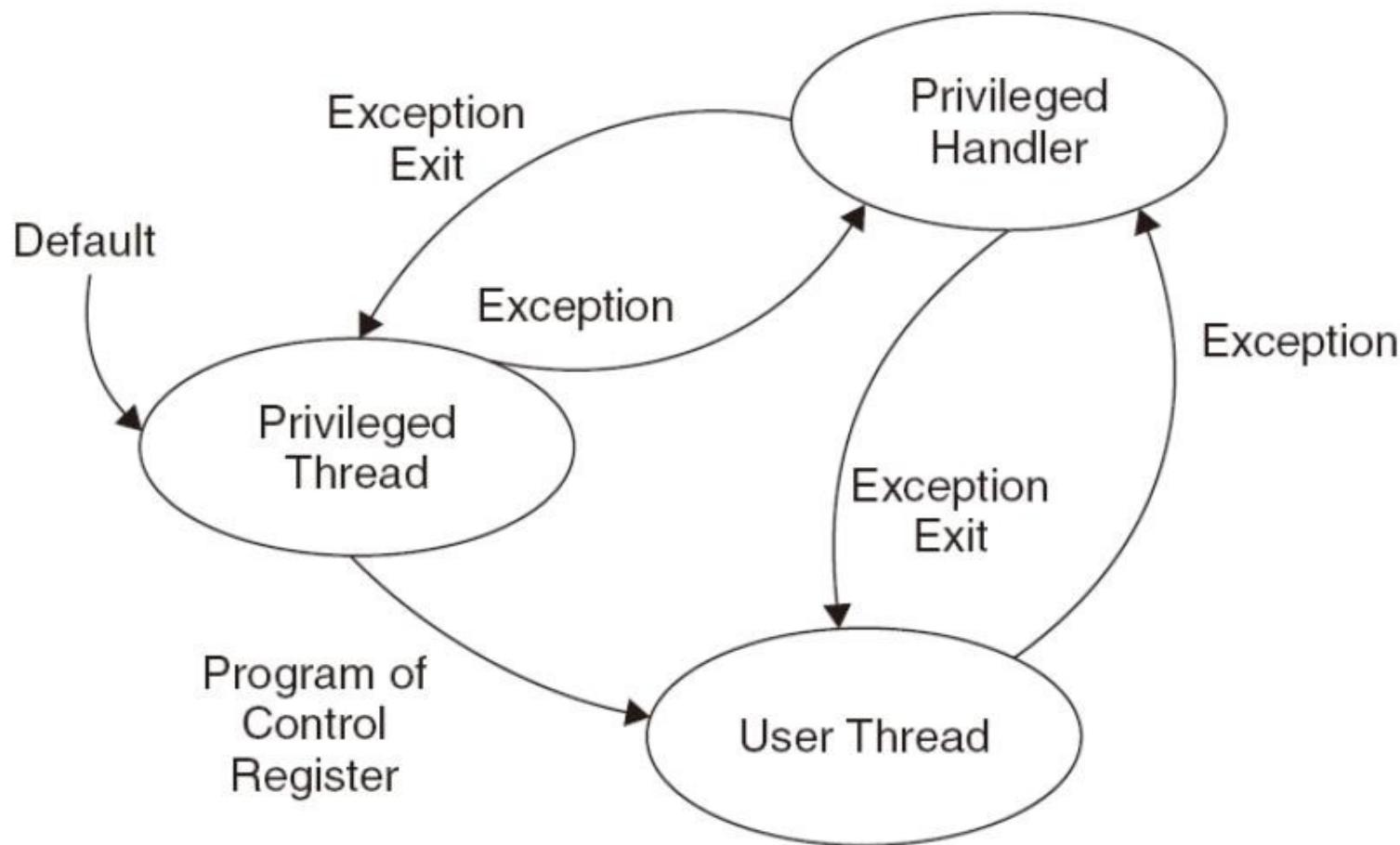


## Advantages:

- 1) No state switching overhead, saving both execution time and instruction space
- 2) No need to separate ARM code and Thumb code source files, making software development and maintenance easier
- 3) easier to write software because there is no need to worry about switching code



- Two modes and two privilege levels





## 8.3 Cortex-M3 Basics

### 8.3.1 Registers

general registers

stack register

link register

program counter

special registers

Name	Functions (and Banked Registers)
R0	General-Purpose Register
R1	General-Purpose Register
R2	General-Purpose Register
R3	General-Purpose Register
R4	General-Purpose Register
R5	General-Purpose Register
R6	General-Purpose Register
R7	General-Purpose Register
R8	General-Purpose Register
R9	General-Purpose Register
R10	General-Purpose Register
R11	General-Purpose Register
R12	General-Purpose Register
R13(MSP)	Main Stack Pointer(MSP), Process Stack Pointer(PSP)
R14	Link Register(LR)
R15	Program Counter(PC)



### 8.3.1.1 General-Purpose Registers

#### 1. **R0~R7** (*low registers*):

Accessed by all 16-bit Thumb instructions and all 32-bit Thumb-2 instructions; reset value is unpredictable

#### 2. **R8~R12** (*high registers*):

Accessible by all Thumb-2 instructions but not by all 16-bit Thumb instructions; reset value is unpredictable

### 8.3.1.2 Stack Pointers

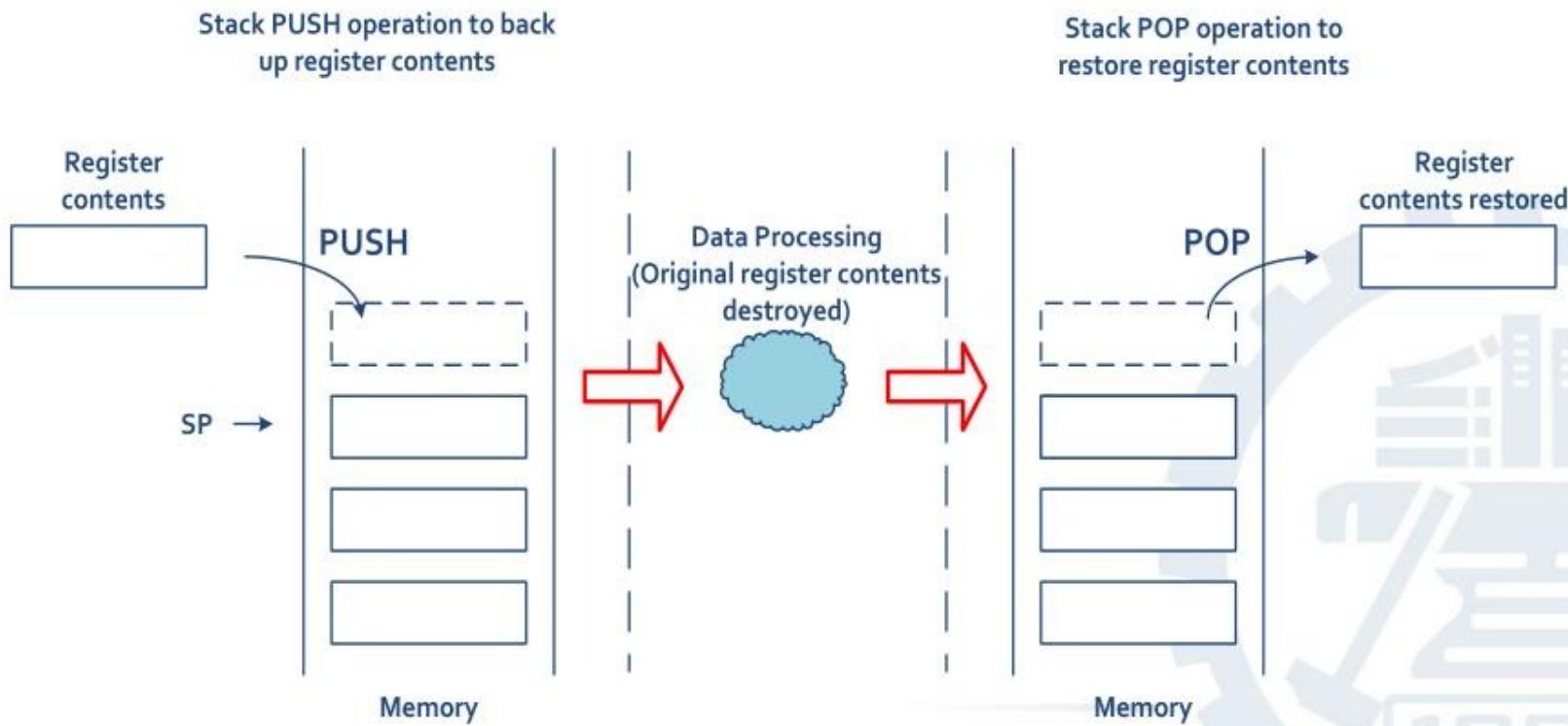
**R13** is the stack pointer. Two stack pointers are banked so that only one is visible at a time.

The two stack pointers are:

1. **Main Stack Pointer (MSP)** : This is the default stack pointer, used by the OS kernel, exception handlers, and privileged-mode programs
2. **Process Stack Pointer (PSP)** : Used by the user application code

## Stack: a first-in last-out buffer

- The stack pointers are used for accessing stack memory processes such as **PUSH** and **POP**
- Used to store register contents to stack memory at the start of a subroutine and then restore the registers from stack at the end of the subroutine.



## More about stack in Cortex-M3

- Cortex-M3 uses a full-descending stack arrangement

The stack pointer decrements when new data is pushed in the stack

- The assembly language syntax is as follows:

PUSH {R0} ; R13=R13-4, then Memory[R13]=R0

POP {R0} ; R0=Memory[R13], then R13=R13+4

- Either **R13** or **SP** can be used in your program codes  
(referring to the MSP/PSP)

- A particular stack pointer (**MSP/PSP**) can be accessed  
using special register access instructions (**MRS/MSR**)



## More about stack in Cortex-M3

- Since PUSH and POP operations are always word aligned, the stack pointer R13 bit 0 and bit 1 are hardwired to zero and always read as zero (RAZ)





### 8.3.1.3 Link Register

**R14** is the link register (**LR**). LR is used to store the return program counter when a subroutine or function is called.

e.g., when using the **BL** (Branch with Link) instruction:

main ; Main program

...

BL function1 ; Call function1 using Branch with Link instruction:  
; PC = function1  
; LR = the next instruction in main

...

function1

...

BX LR ; Return

; Program code for function 1

### 8.3.1.4 Program Counter

**R15** is the program counter (**PC**).

When you read this register you will find that the value is different than the location of the executing instruction by 4, due to the pipelining of the processor



**Example:**

0x1000 : MOV R0, PC ; R0 = 0x1004

When reading the **PC**, the LSB (bit 0) is always 0.

**Why?**

When writing to the **PC**, it will cause a branch. (*Can you write to the PC in 8086?*); the LSB must be set to 1 to indicate the Thumb state operations (setting to 0 implies to switch to the ARM state, which will result in a fault exception in Cortex-M3)

## 8.3.2 Special Registers

- The special registers in the Cortex-M3 processor include:
  - ❖ Program Status Registers (PSRs)
  - ❖ Interrupt Mask Registers (PRIMASK, FAULTMASK, and BASEPRI)
  - ❖ Control Register (CONTROL)
- Can only be accessed via **MSR** and **MRS** instructions

E.g.,

MRS <reg>, <special\_reg> ; Read special register

MSR <special\_reg>, <reg> ; write to special register

**Note:** MSR and MRS cannot have memory addresses, only registers are allowed

### 8.3.2.1 Program Status Registers (PSRs)

- The program status registers are subdivided into three status registers:

  1. Application PSR (**APSR**), 2. Interrupt PSR (**IPSR**), 3. Execution PSR (**EPSR**)

	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					

- When they are accessed as a collective item, the name xPSR is used (**PSR** used in program codes).

	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				



**EPSR** and **IPSR** are **read-only**:

MRS r0, APSR	; Read Flag state into R0
MRS r0, IPSR	; Read Exception/Interrupt state
MRS r0, EPSR	; Read Execution state
MSR APSR, r0	; Write Flag state

Accessing xPSR:

MRS r0, PSR	; Read the combined program status word
MSR PSR, r0	; Write combined program state word



## Bit Fields in Cortex-M3 Program Status Registers

Bit	Description
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception Number	Indicates which exception the processor is handling

## 8.3.2.2 PRIMASK, FAULTMASK and BASEPRI Registers

The PRIMASK, FAULTMASK, and BASEPRI registers are used to disable exceptions.

### Cortex-M3 Interrupt Mask Registers

Register Name	Description
PRIMASK	A 1-bit register. When this is set, it allows NMI and the hard fault exception; all other interrupts and exceptions are masked; default is 0 (no masking)
FAULTMASK	A 1-bit register. When this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled; default is 0
BASEPRI	A register of up to 9 bits. It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value); default is 0

- To access the PRIMASK, FAULTMASK, and BASEPRI registers, the MRS and MSR instructions are used.



### Example:

```
MRS    r0, BASEPRI      ; Read BASEPRI register into R0
MRS    r0, PRIMASK      ; Read PRIMASK register into R0
MRS    r0, FAULTMASK    ; Read FAULTMASK register into R0
MSR    BASEPRI, r0      ; Write R0 into BASEPRI register
MSR    PRIMASK, r0      ; Write R0 into PRIMASK register
MSR    FAULTMASK, r0    ; Write R0 into FAULTMASK register
```

- PRIMASK and BASEPRI are useful for temporarily disabling interrupts in timing-critical tasks; FAULTMASK is used by the OS kernel which cleans up a crashed task
- The PRIMASK, FAULTMASK, and BASEPRI registers cannot be set in the user access level.

### 8.3.2.3 The Control Register

The **Control register** is used to define the privilege level and the stack pointer selection. This register has two bits.

Bit	Function
CONTROL[1]	<p>Stack status:</p> <p>1 = Alternate stack is used</p> <p>0 = Default stack (MSP) is used</p> <p>If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.</p>
CONTROL[0]	<p>0 = Privileged in Thread mode</p> <p>1 = User state in Thread mode</p> <p>If in handler mode (not Thread mode), the processor operates in privileged mode.</p>





## ● CONTROL[1]

In Cortex-M3, the CONTROL[1] bit is always 0 (MSP) in handler mode. However, in the Thread mode, it can be either 0 or 1.

This bit is writable only when the core is in Thread mode and privileged.

## ● CONTROL[0]

The CONTROL[0] bit is writable only in privileged level.

To access the Control register, the MRS and MSR instructions are used:

MRS r0, CONTROL ; Read CONTROL register into R0

MSR CONTROL, r0 ; Write R0 into CONTROL register

### 8.3.3 Operation Mode

Two modes and two privilege levels.

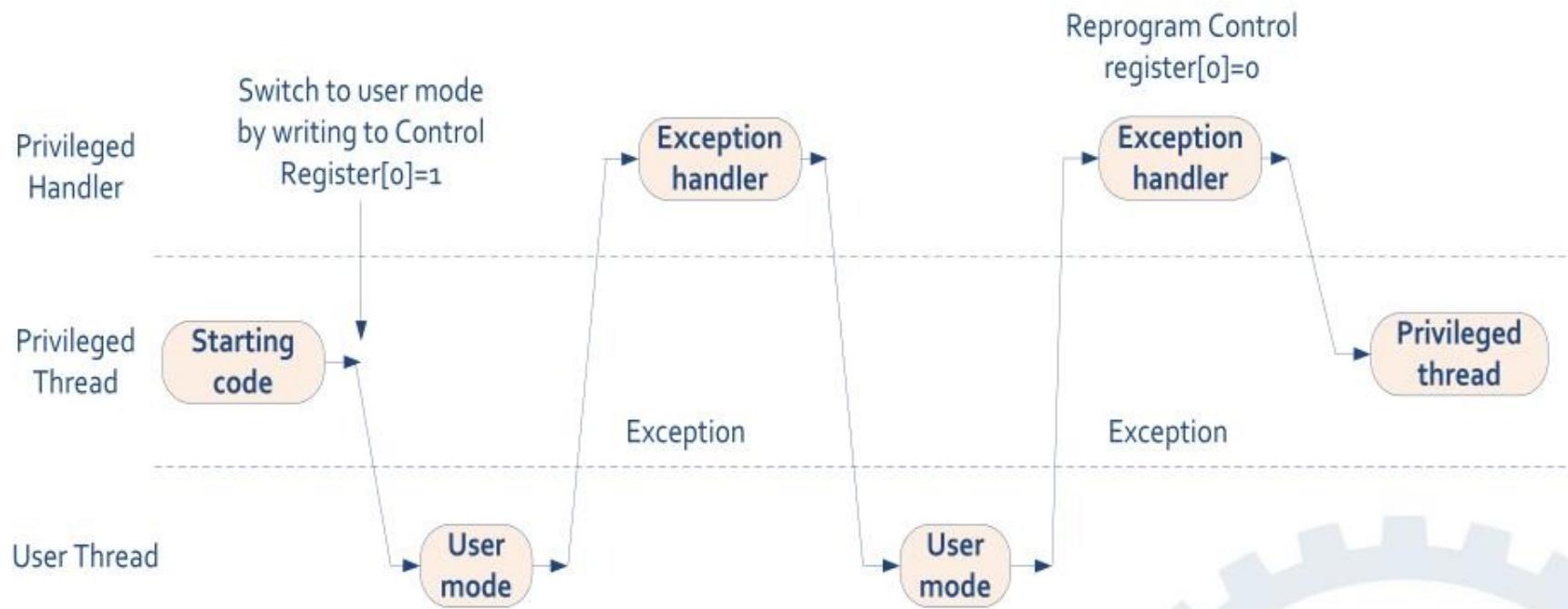
**Operation Modes and Privilege Levels in Cortex-M3**

	<i>Privileged Level</i>	<i>User Level</i>
When running an exception	<i>Handler Mode</i>	
When running main program	<i>Thread Mode</i>	<i>Thread Mode</i>

The operation modes determine whether the processor is running a normal program or running an exception handler.

- When the processor exits reset, it is in Thread mode with privileged access level
- In the user access level (Thread mode), access to the System Control Space (SCS, a memory block for configuration registers and debugging components) is blocked.
- In the user access level, access to special registers (except the **APSR**) is also blocked. Fault exception will occur when trying to access SCS or special registers
- Software in a privileged access level can switch the program into the user access level by setting CONTROL[0]=1
- When an exception occurs, the processor will automatically switch to privileged state and return to the previous state when exiting the exception handler

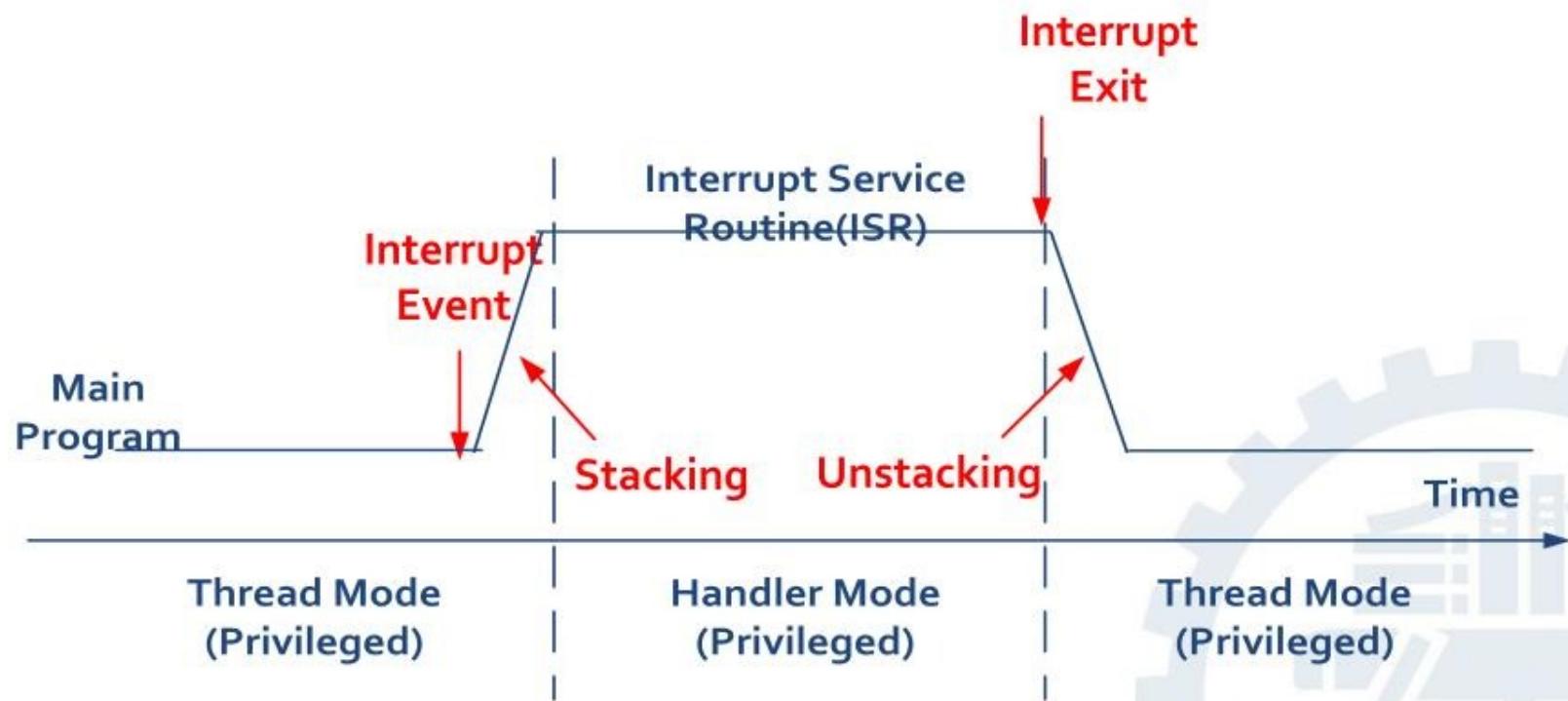
Therefore, in order to change to privileged level, a user program has to go through an exception/interrupt handler which can set the CONTROL register before returns.



## Switching of Operation Mode by Programming the Control Register or by Exceptions



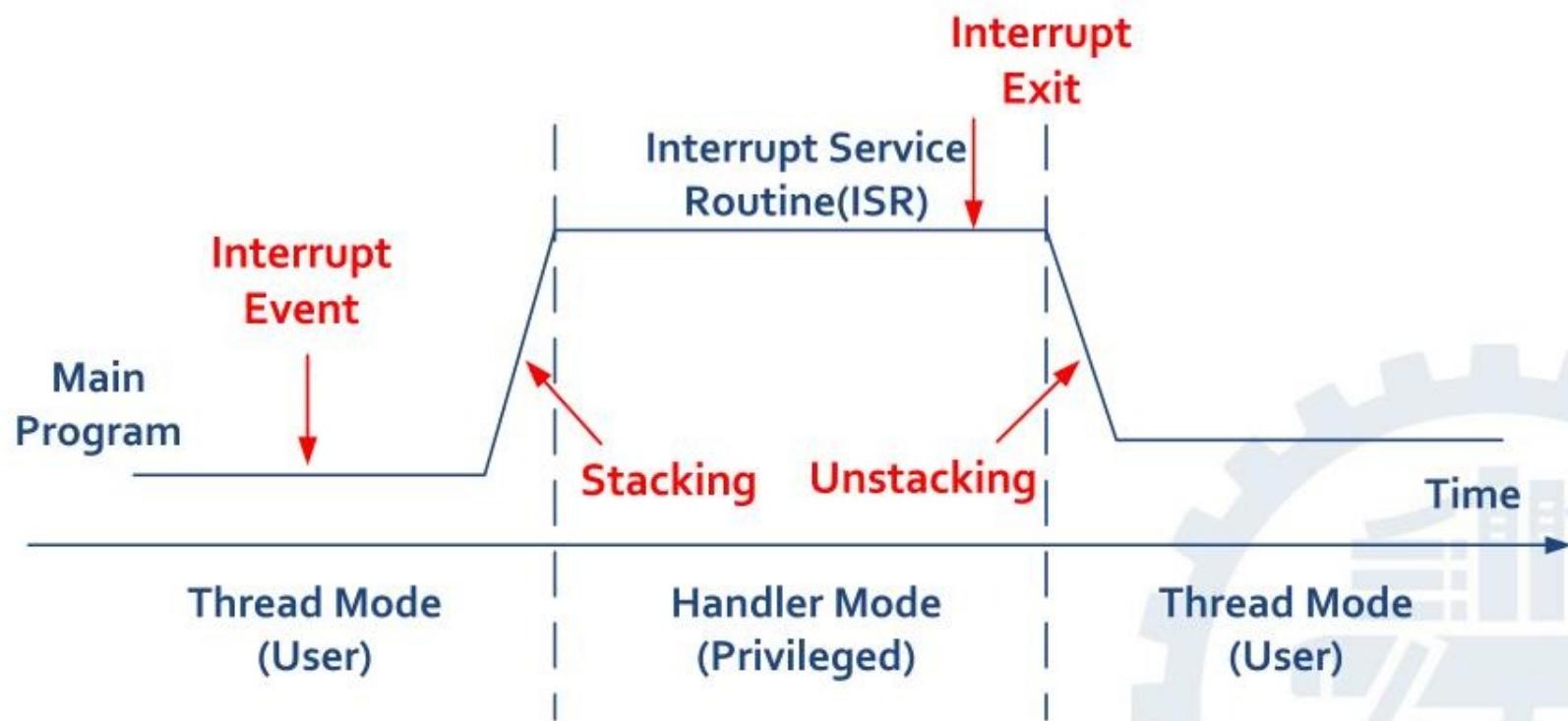
When the Control register[0] = 0, only the processor mode changes when an exception takes place.



Switching Processor Mode at Interrupt



**When the Control register[0] = 1, both processor mode and access level change when an exception takes place.**



**Switching Processor Mode and Privilege Level at Interrupt**

### 8.3.4 Vector Tables

- The **vector table** is an array of word data, with each representing the starting address of the ISR for one exception/interrupt type.
- The base address of the vector table is relocatable (set the relocation register in the NVIC); initially, the base address is 0x0.



#### Example:

The reset is exception type 1. The address of the reset vector is 1 times 4, which equals 0x00000004; and NMI vector (type 2) is located in  $2 * 4 = 0x00000008$

The word stored at address 0x00000000 is used as the starting value for the MSP.



## Vector Table Definition After Reset

Exception Type	Address Offset	Exception Vector
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug Monitor
11	0x2C	SVC
7-10	0x1C-0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

## 8.3.5 Stack Memory Operations

### 8.3.5.1 Basic Operations of the Stack

- Multiple registers can be pushed and popped in one instruction:

PUSH *reglist* ; push the large numbered registers first

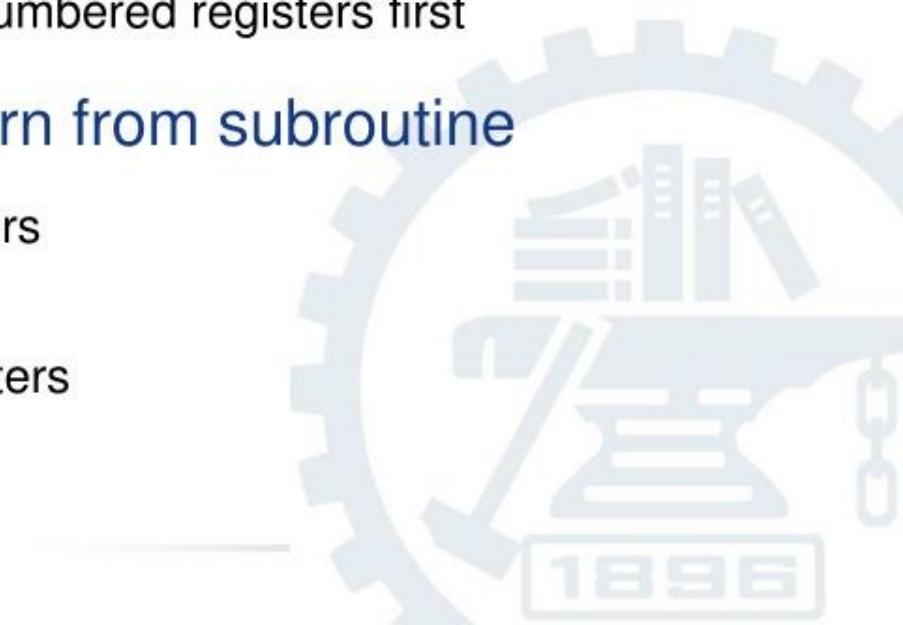
POP *reglist* ; pop the low numbered registers first

- Using PUSH and POP to return from subroutine

PUSH {R0-R7, LR} ; Save registers

...

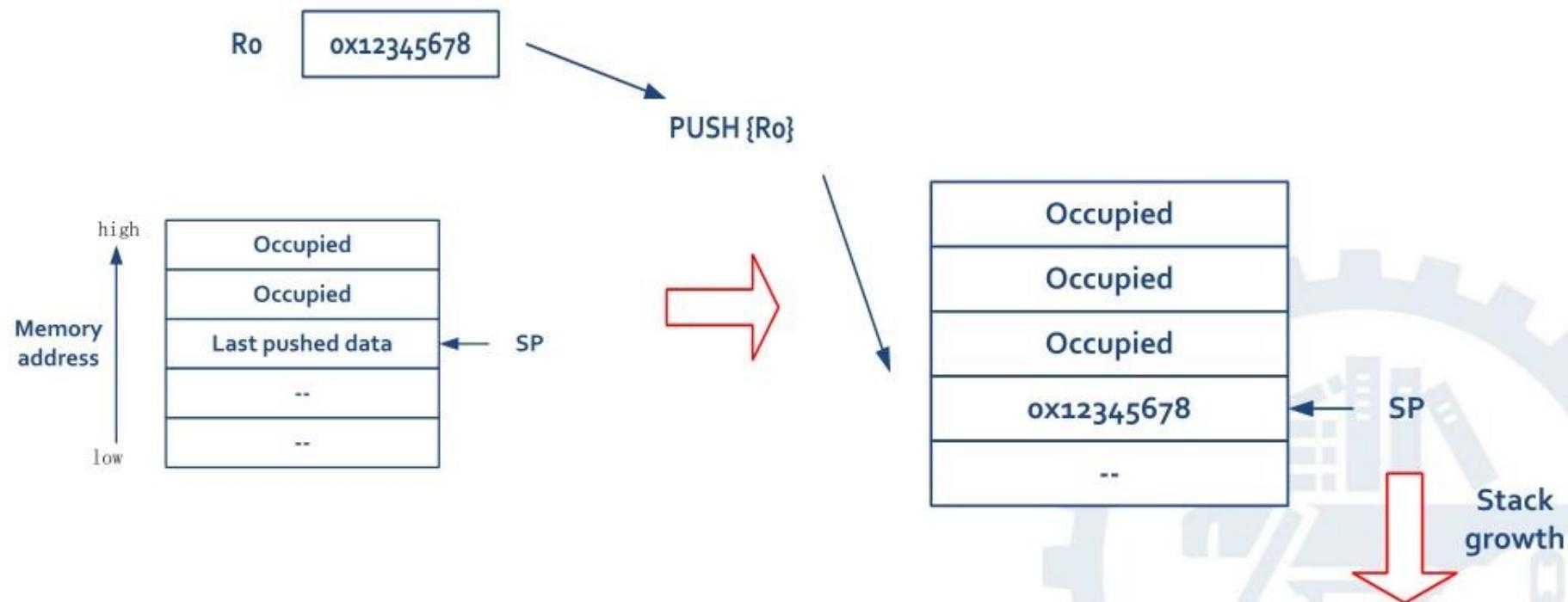
POP {R0-R7, PC} ; Restore registers





### 8.3.5.2 Cortex-M3 Stack Implementation

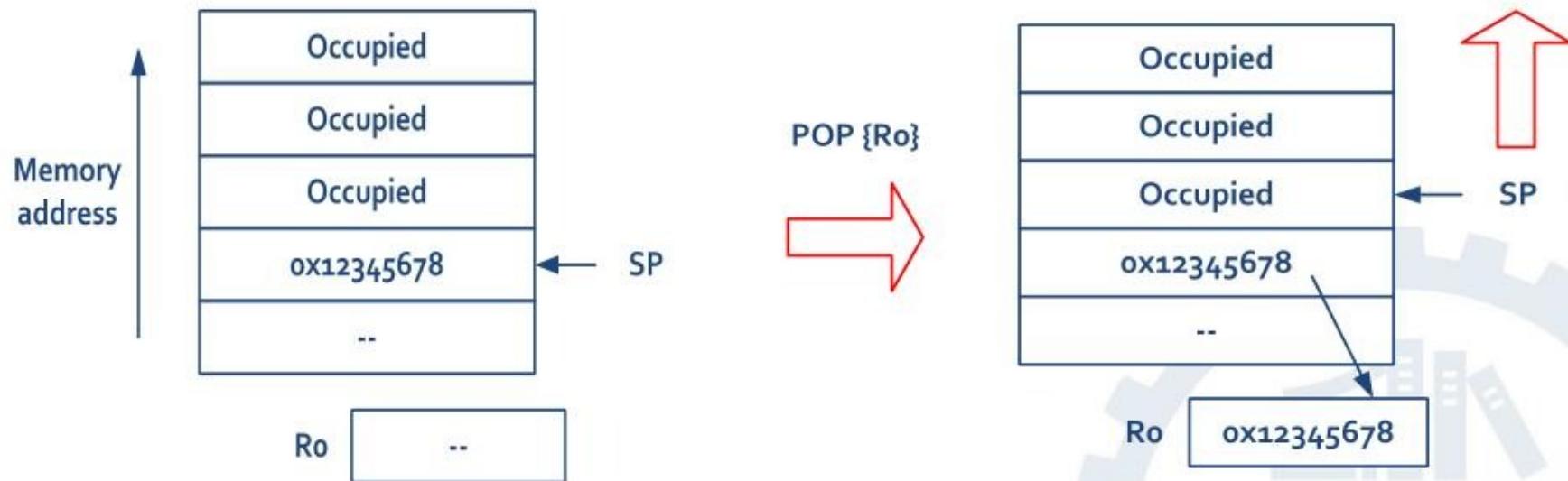
The stack pointer (SP) points to the last data pushed to the stack memory, and the SP decrements first in a new **PUSH** operation.



Cortex-M3 Stack PUSH Implementation



For POP operations, the data is read from the memory location pointer by SP, then the stack pointer is incremented. The contents in the memory location are unchanged.

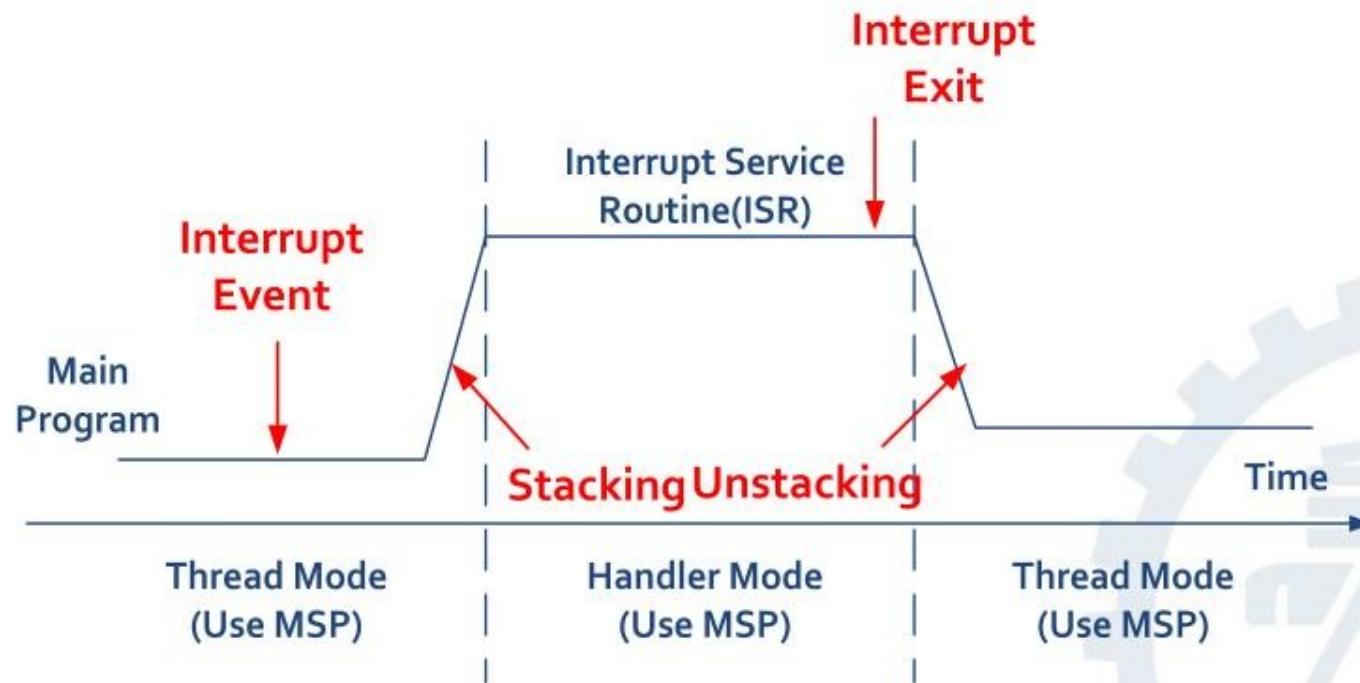


## Cortex-M3 Stack POP Implementation

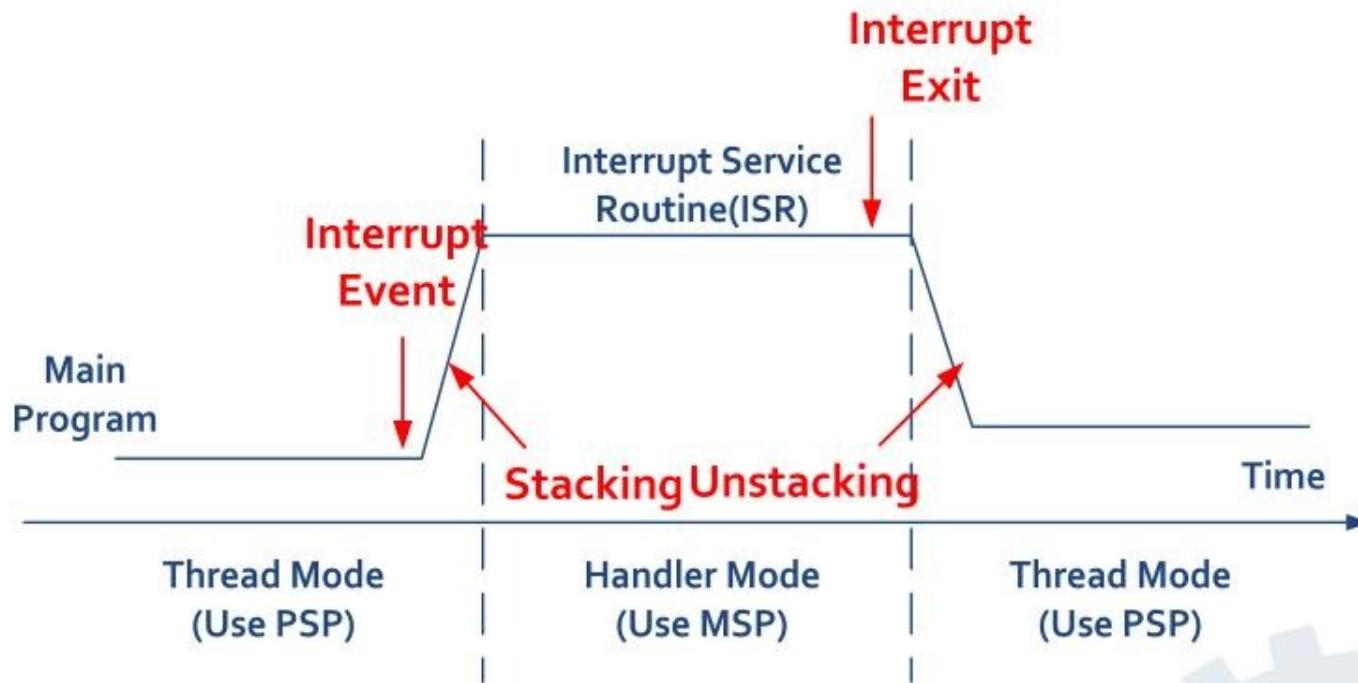


### 8.3.5.3 The Two-Stack Model in the Cortex-M3

The Cortex-M3 has two stack pointers: the Main Stack Pointer (MSP) and the Process Stack Pointer (PSP). The SP register to be used is controlled by the Control register bit 1.



**Control [1] = 0: Both Thread Level and Handler Use Main Stack**



**Control[1] = 1: Thread Level Uses Process Stack and Handler Uses Main Stack**

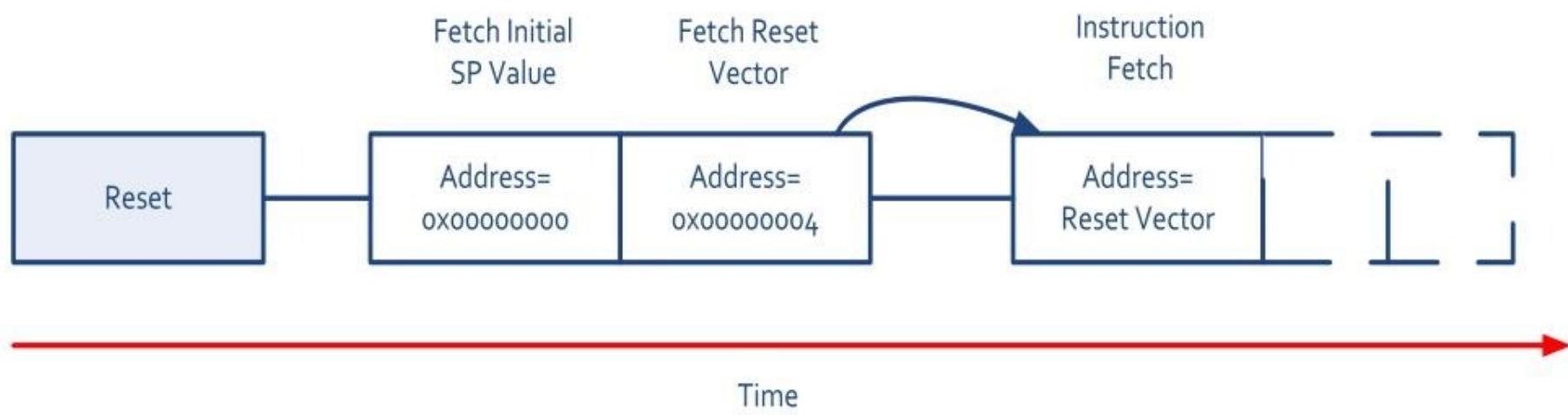
The automatic stacking and unstacking mechanism will use PSP; whereas stack operations inside the handler will use MSP.



### 8.3.5.4 Reset Sequence

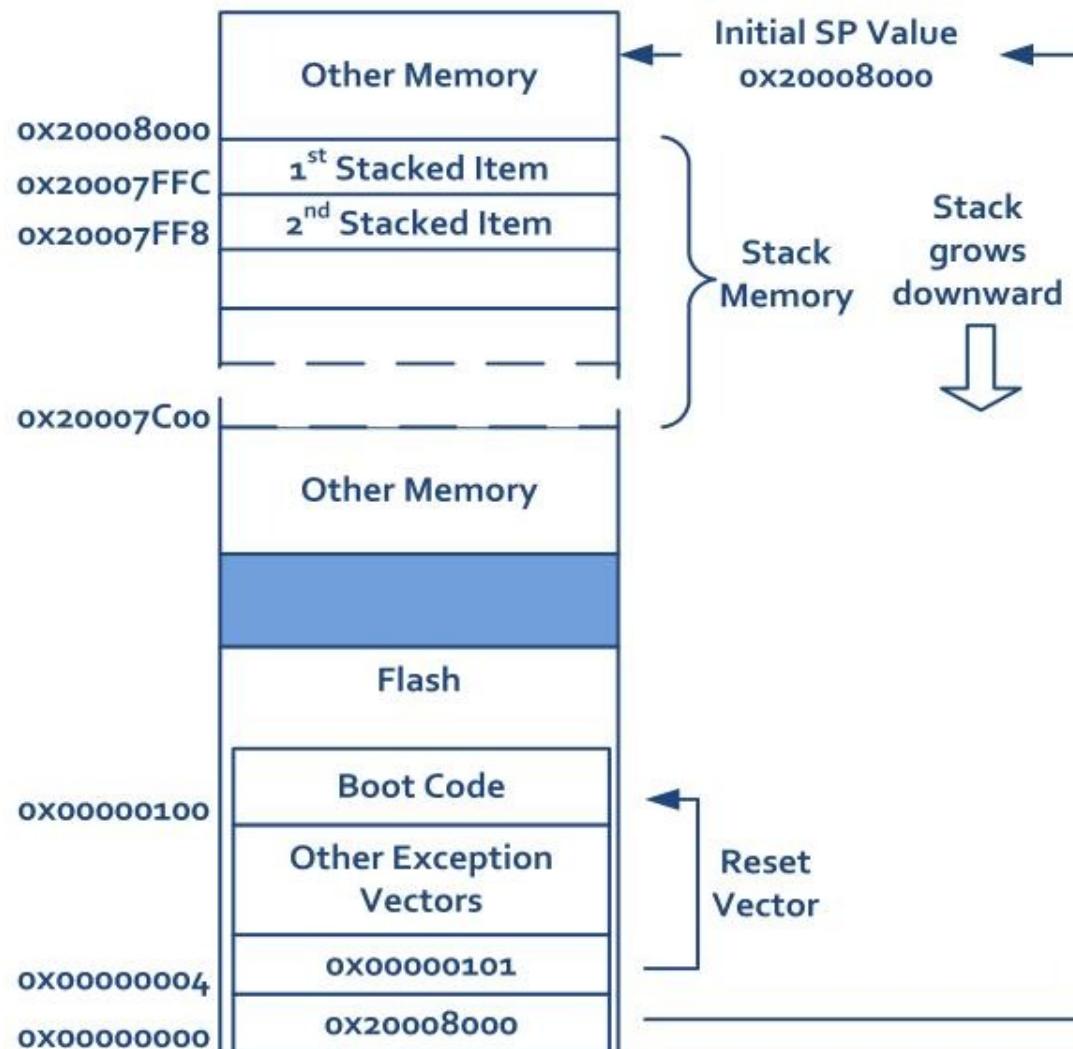
After the processor exits reset, it will read two words from memory:

1. Address 0x00000000: default value of R13 (MSP)
2. Address 0x00000004: Reset vector (the starting address of startup program; LSB should be set to 1 to indicate Thumb state)



## Reset Sequence

The vector table starts after the initial SP value. In the Cortex-M3, vector addresses in the vector table should have their LSB set to 1 to indicate that they are Thumb code.



Assume that the stack memory range from 0x20007C00 to 0x20007FFF

## Initial Stack Pointer Value and Initial Program Counter (PC) Value Example